# Tiva Based Daughter Board for Firebird V Hardware And Software Manual.

**eRTS Lab IIT Bombay**

June 27, 2017

# 1 Credits

**Version 1.0**
**June 27, 2017**

**Documentation Author(Alphabetical Order):**

1. Ayush Gaurav, Intern eYSIP 2017
2. Nagesh K, Intern eYSIP 2017

**Credits(Alphabetical Order):**

1. Prof Kavi Arya, CSE IIT Bombay
2. Nex Robotics Pvt. Ltd.
3. Piyush Manavar, Team e-Yantra
4. Saurav Shandilya, Team e-Yantra

# 2 Notice

The contents of this manual are subject to change without notice. All efforts have been made to ensure the accuracy of contents in this manual. However, should any errors be detected, e-yantra welcomes your corrections. You can send us your queries / suggestions at Contact Us

# Table Of Content

# 3 Introduction

Tiva Daughter board for Fire Bird V will help you gain exposure to the world of robotics and embedded systems with ARM Cortex M4. The board is designed with Open Source Philosophy in software and hardware design ,you will be able to create and contribute to complex applications that run on this platform, helping you acquire expertise as you spend more time with them.

### 3.1 Safety precautions:

- Robot's electronics is static sensitive. Use robot in static free environment.
- Read the assembling and operating instructions before working with the robot.
- If robot's battery low buzzer starts beeping, immediately charge the batteries.
- To prevent fire hazard, do not expose the equipment to rain or moisture.
- Refrain from dismantling the unit or any of its accessories once robot is assembled.
- Charge the NiMH battery only with the charger provided on the robot.
- Never allow NiMH battery to deep discharge.
- Mount all the components with correct polarity.
- Keep wheels away from long hair or fur.
- Keep the robot away from the wet areas. Contact with water will damage the robot.
- To avoid risk of fall, keep your robot in a stable position.
- Do not attach any connectors while robot is powered ON.
- Never leave the robot powered ON when it is not in use.
- Disconnect the battery charger after charging the robot.

### 3.2 Inappropriate Operation:

Inappropriate operation can damage your robot. Inappropriate operation includes, but is not limited to:

- Dropping the robot, running it off an edge, or otherwise operating it in irresponsible manner.
- Interfacing new hardware without considering compatibility.
- Overloading the robot above its payload capacity.
- Exposing the robot to wet environments.
- Continuing to run the robot after hair, yarn, string, or any other item is entangled in the robot's axles or wheels.
- All other forms of inappropriate operations.
- Using robot in areas prone to static electricity.
- Read carefully paragraphs marked with caution symbol.

# 4 Tiva Based Daughter Board

There are two daughter boards one with the launchpad and other one with the Arm Cortex M4 based uC. Almost all the specification are same unless mentioned otherwise.

## 4.1 Technical Specification

**Microcontroller:**
TM4C123gh6pm (ARM architecture based Microcontroller)
To know more about the microcontroller please refer to datasheet.

**Sensors:**
Three white line sensors (extendable to 7)
Five Sharp GP2Y0A02YK IR range sensor (One in default configuration)
Eight analog IR proximity sensors
Two position encoders

**Indicators:**
2 x 16 Characters LCD
Buzzer

**Communication:**
USB Communication
Wireless ZigBee Communication (2.4GHZ) (if XBee wireless module is installed)
Bluetooth communication(Can be interfaced on external UART0 available on the board)
Simplex infrared communication (From infrared remote to robot)
I2C Communication

**Battery Life:**
2 Hours, while motors are operational at 75% of time

**Locomotion:**
Two DC geared motors in differential drive configuration and caster wheel at front as support
Top Speed: 24 cm / second
Wheel Diameter: 51mm
Position encoder: 30 pulses per revolution
Position encoder resolution: 5.44 mm

# 5 Hardware Manual:

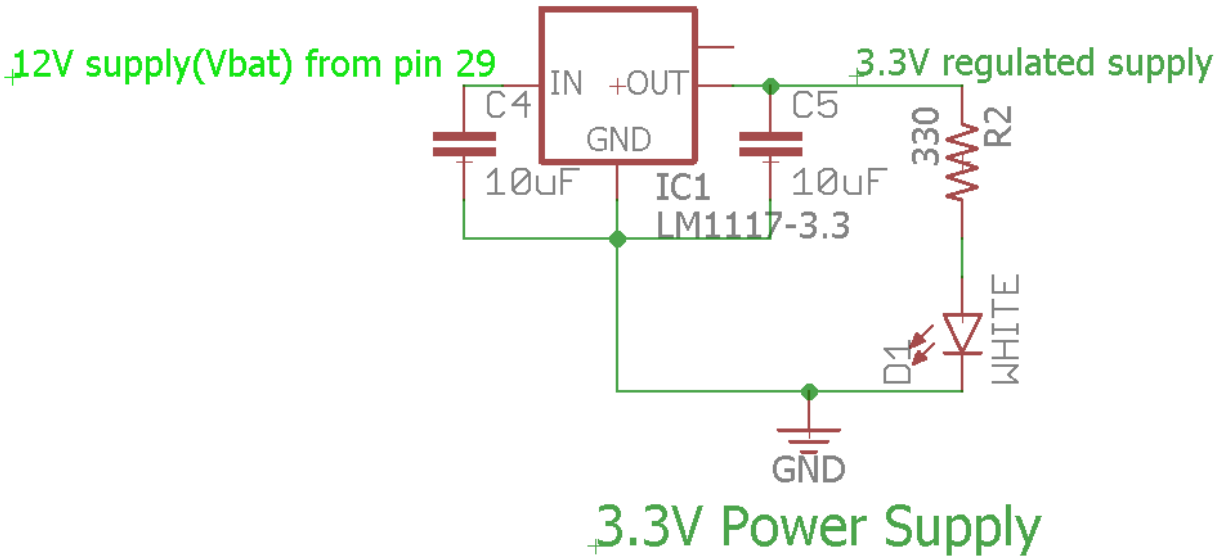
## 5.1 Voltage Regulation on the Daughter Board

The voltage source available on the Firebird is 9.6V. But the TIVA platform works on 3.3V and the servos can operate upto 6V. So there must be 3 different voltage levels on the board. The uC based board has 2 voltage regulators and the plug and play board has 1 voltage regulator. In the uC based board the

9.6 volts is 3.3V to power the microcontroller. In the plug and play board the there is an inbuilt voltage regulator, so it is directly connected connected to 5v, 300mA source. The servo in both the boards has a separate 5V regulator.

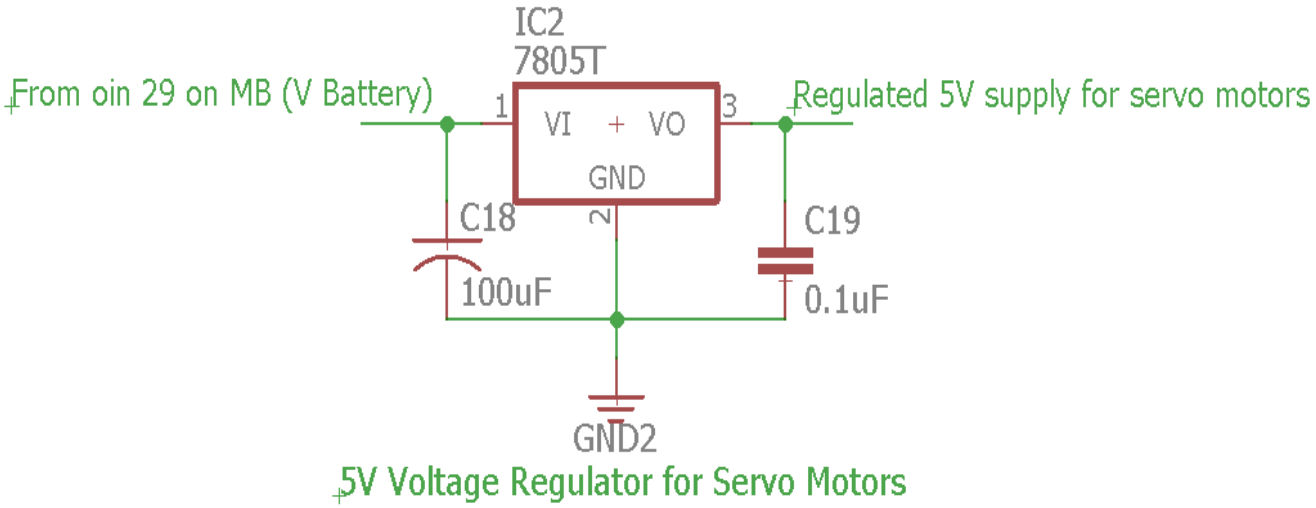
### 5.1.1 Powering Micro-controller

The boards have different powering circuits.In the plug and play board is connected to 5V source on Pin 10. In the uC based board the 9.6V source available on Pin 29 is reduced to 3.3V. Refer to the schematic below for further details.



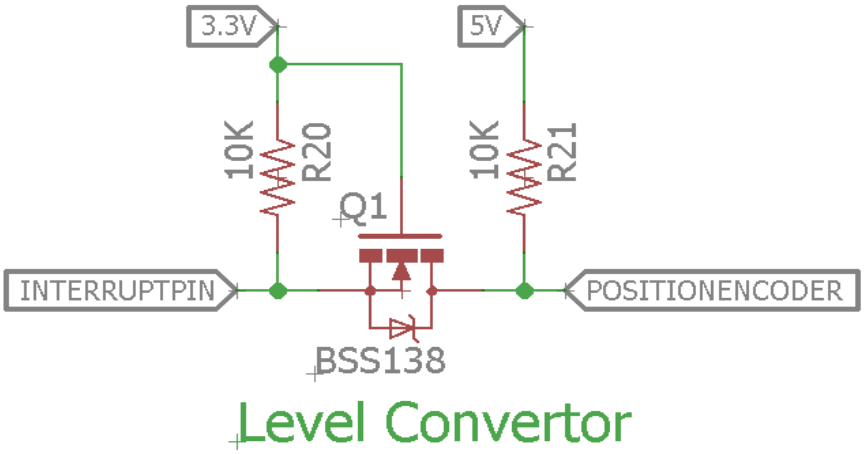

### 5.1.2 Powering Servos

The servo motors can operate safely up to 6V, beyond this voltage they get damaged. Also, the servos require high current. There is a separate power line for servos taken from Pin 29 and reduced to 5V using the voltage regulator. Refer the schematic for further details.



## 5.2 Level Converters

The TIVA platform operates at 3.3V and the Firebird operates at 5V. Directly connecting these pins to the TIVA may be fatal. So to interface these sensors, a level converter is used. A bidirectional MOSFET based level converter used. The level converter is necessary is for input pins. In the boards Level converter is used for interfacing the position encoders of the motors. Refer the schematic for further details.




**NOTE: If the user wishes to interface extra sensors using the GPIOs provided on the board, then external level converters have to used if the output of the sensor is above 3.3V.**

## 5.3 Sensors

The firebird V has as many as 22 sensors, but maximum 12 sensors can be interfaced directly with the controller. The daughter board has interfaced 20 of those 22 sensors using external I2C bases ADC. Sensors that were not included in the daughter board are current sensor and battery monitoring sensor. These sensors are working either on 3.3V or on 5V. Interfacing 3.3V sensors are simple and can be directly connected to the controller. On the other hand 5V can not be directly interfaced so a different approach is taken which will be mentioned in the 5V sensors sub heading.

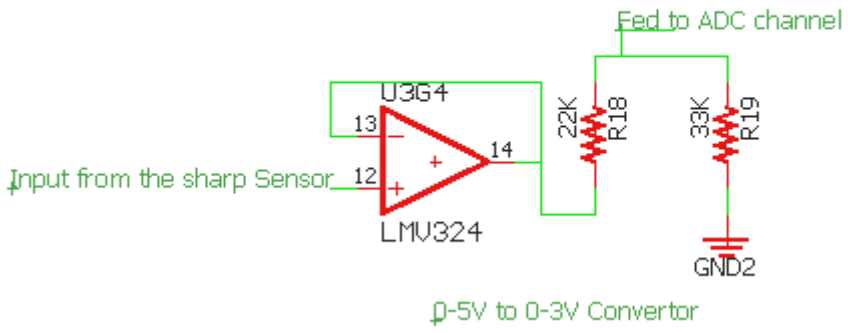
### 5.3.1 3.3V sensors

The output white line sensors and IR Proximity sensors vary from 0 to 3.3V. Hence these sensors can be interfaced directly with the microcontroller. Refer the table below for pin connections.

| IR Proximity Sensors | Pin Name(uC) | Pin Name(Plug and Play) |
|---|---|---|
| 1 | PE1 | PB5 |
| 2 | PE3 | PD0 |
| 3 | PE5 | PD3 |
| 4 | PE4 | PD1 |
| 5 | PB5 | PE5 |
| 6 | External ADC IN6 | External ADC IN7 |
| 7 | External ADC IN7 | PE0 |
| 8 | External ADC IN0 | External ADC IN0 |
| | | |
| **White Line Sensors** | **Pin Name(uC)** | **Pin Name(Plug and Play)** |
| 1 | PD2 | PE1 |
| 2 | PD1 | PE2 |
| 3 | PD0 | PE3 |
| 4 | External ADC IN1 | External ADC IN2 |
| 5 | External ADC IN2 | External ADC IN3 |
| 6 | External ADC IN3 | External ADC IN4 |
| 7 | External ADC IN4 | External ADC IN5 |

### 5.3.2 5V sensors

Sharp Sensors are the only sensors on board that works on 5V supply. The output of the sharp sensor ranges from 0-5V and according to the output we have a formula to calculate the distance. While uC has VREF as 3.3V so these sensors cannot be directly connected. The approach we followed is to feed the output of the sensor to a buffer and then using a voltage divider convert 0-5 range to 0-3V range. For better understanding refer to the schematic below. There is a also table which tells about the pin connection.

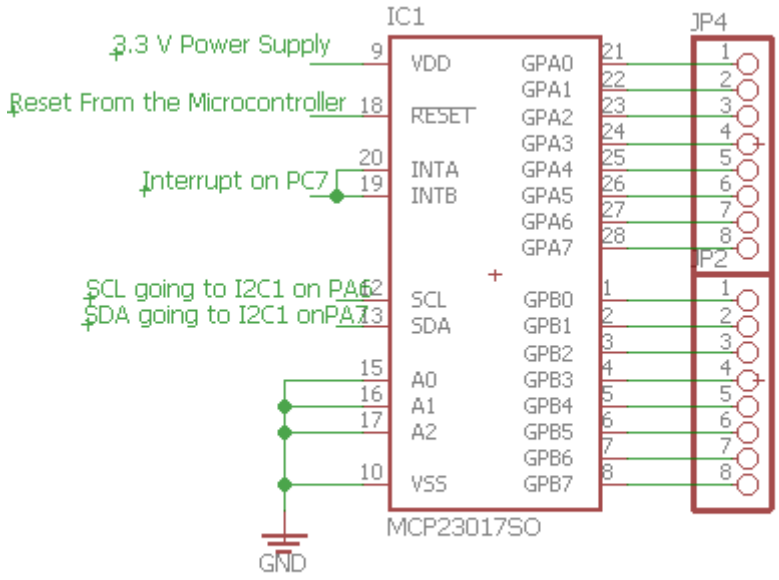



| Sharp Sensors | Pin Name(uC) | Pin Name(Plug and Play) |
|---|---|---|
| 1 | PE0 | PB4 |
| 2 | PE2 | External ADC IN1 |
| 3 | PD3 | PD2 |
| 4 | External ADC IN5 | External ADC IN6 |
| 5 | PB4 | PE4 |

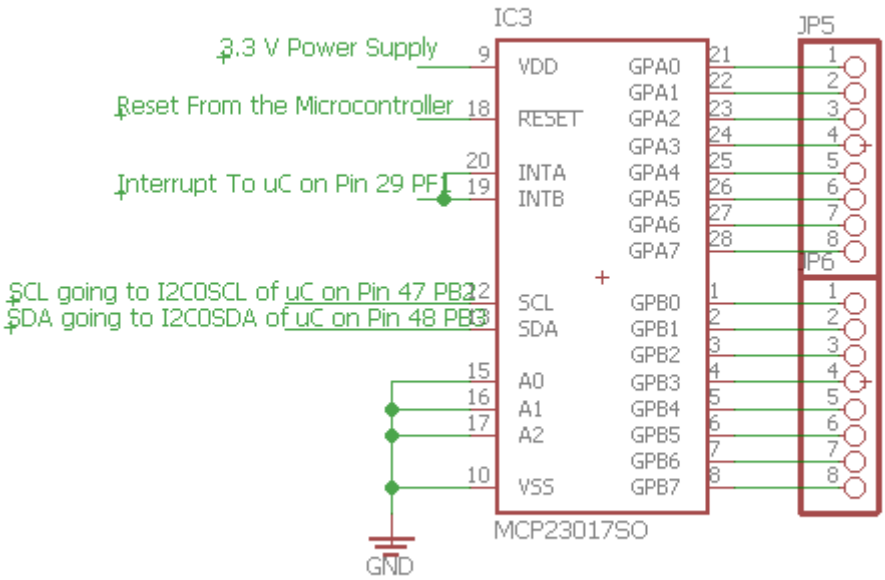
## 5.4 Port Expander

TM4C123GH6PM has only 64 pins out which only 43 are GPIO pins. This limits our application to

read input and respond correspondingly. To increase the number of GPIO and there interrupts we have used I2C compatible a port expander MCP23017. It has 2 PORTS A and B, with each port having 8 Pins.The interrupts on each pin can also be monitored. To read more about it, download the datasheet from here.The schematic of the connection is shown below.Keep in mind that I2C SCL and SDA have already been pulled up using 10K resistor.




Port Expander For Plug and Play

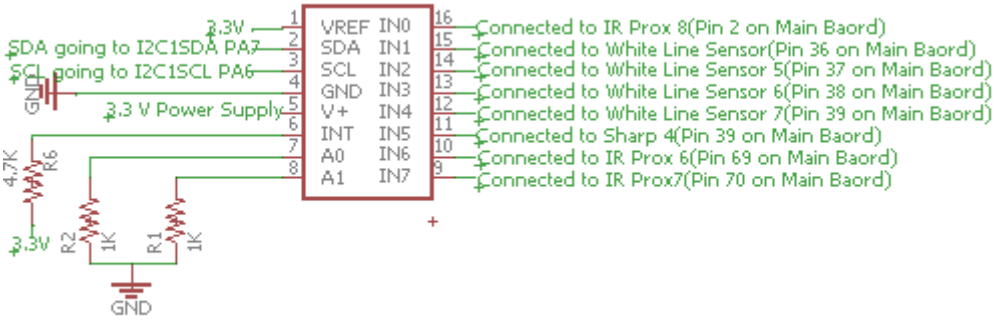



Port Expander For uC

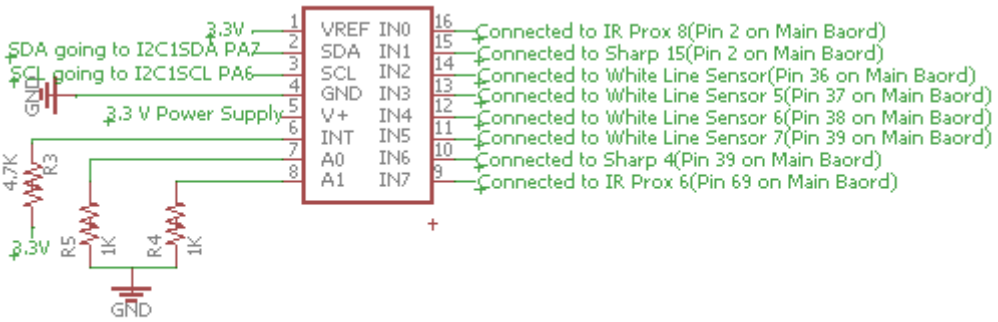
## 5.5 External ADC

It has already been mentioned that adc channels on the microcontroller is limited to 12 while firebird has 22 sensors available. We have interfaced an external ADC which is also I2C compatible. It has 8 channel with 12 bit resolution.To read more about it, download the datasheet from here.The schematic of the connection is shown below.Keep in mind that I2C SCL and SDA have already been pulled up using 10K

resistor.



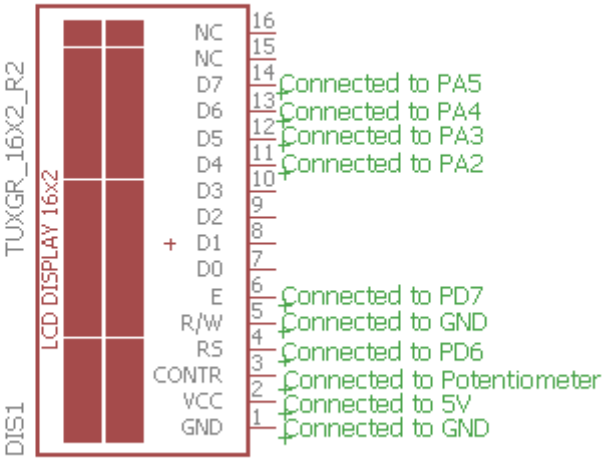

External ADC connection for uC Board



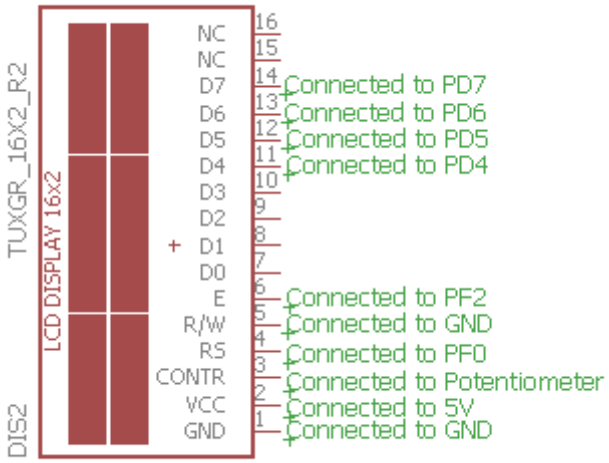

External ADC connections for Plug and Play Board

## 5.6 LCD Interfacing

LCD can be interfaced in 8bit or 4 bit interfacing mode. In 8 bit mode it requires 3 control line and 8 data lines. To reduce number of I/Os required, Fire Bird V robot uses 4 bit interfacing mode which requires 2 control lines and 4 data lines. In this mode upper and lower nibble of the data/command byte needs to be sent separately. RW(Read/Write) control line of lcd is grounded so it can only work in write mode.
The EN line is used to tell the LCD that microcontroller has sent data to it or microcontroller is ready to receive data from LCD. This is indicated by a high-to-low transition on this line. To send data to the LCD, program should make sure that this line is low (0) and then set the other two control lines as required and put data on the data bus. When this is done, make EN high (1) and wait for the minimum amount of time as specified by the LCD datasheet, and end by bringing it to low (0) again.
When RS is low (0), data is treated as a command or special instruction by the LCD (such as clear screen, position cursor, etc.). When RS is high (1), data being sent is treated as text data which should be displayed on the screen.
written to the LCD.//




LCD Connections of Plug and Play board

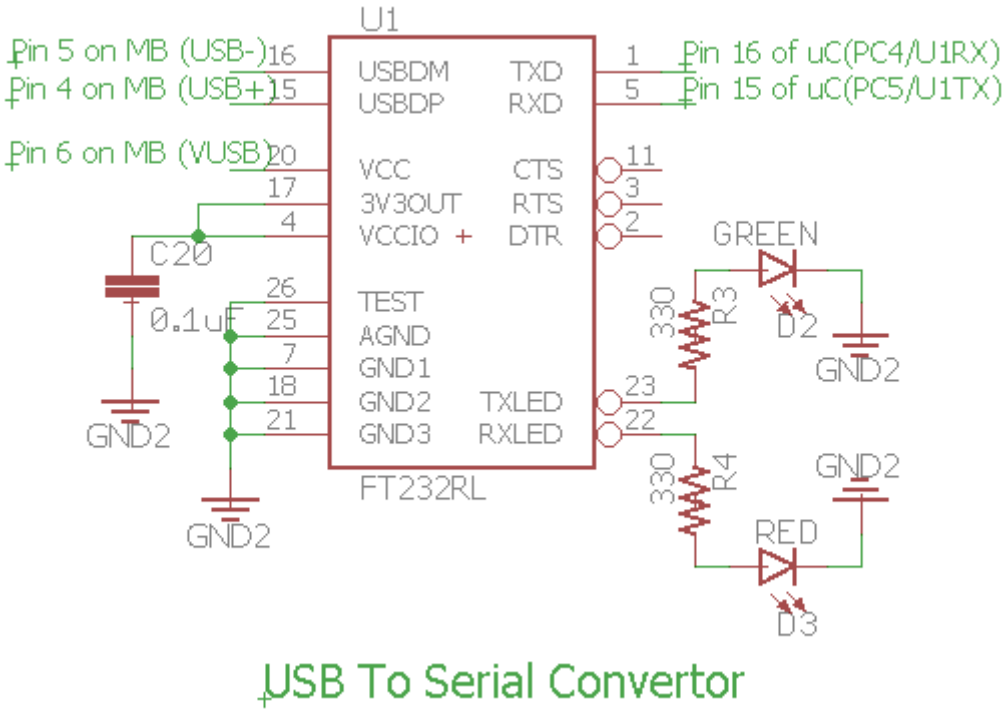



LCD Connections of uC based board

| LCD | Pin Name(uC) | Pin Name(Plug and Play) |
|---|---|---|
| RS | PF0 | PD6 |
| EN | PF2 | PD7 |
| DB4 | PD4 | PA2 |
| DB5 | PD5 | PA3 |
| DB6 | PD6 | PA4 |
| DB7 | PD7 | PA5 |

## 5.7 USB Communication

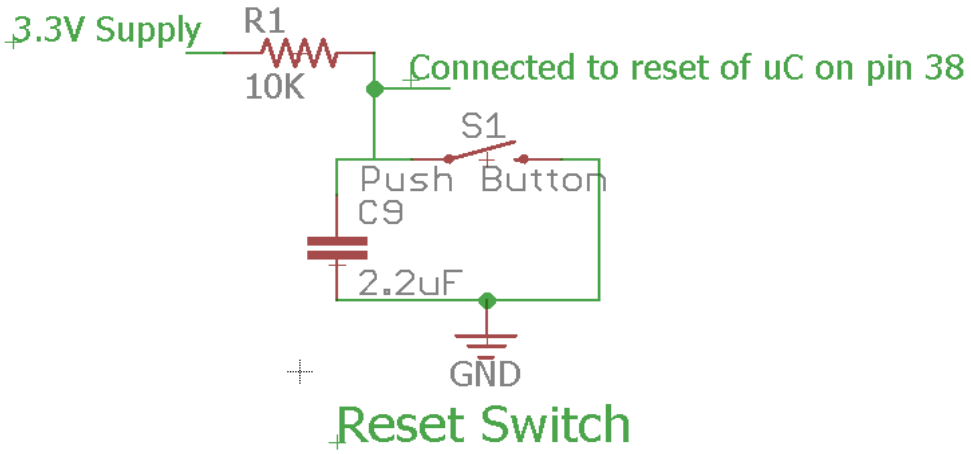
Fire Bird V's main board has USB port socket. Microcontroller accesses USB port via main board socket. All its pins are connected to the microcontroller adapter board via main board's socket connector.FT232 is a USB to TTL level serial converter. It is used for adding USB connectivity to the microcontroller adapter board. With onboard USB circuit Fire Bird V can communicate serially with the PC through USB port without the use of any external USB to Serial converter. Microcontroller socket uses USB port from the main board. Data transmission and reception is indicated suing TX and RX LEDs which are located near the FT232 IC. This IC is only on the uC based board. Plug and play board has its own usb port on TIVA launcpad. The schematic of ft232 is shown below.




## 5.8 Programing the Controller

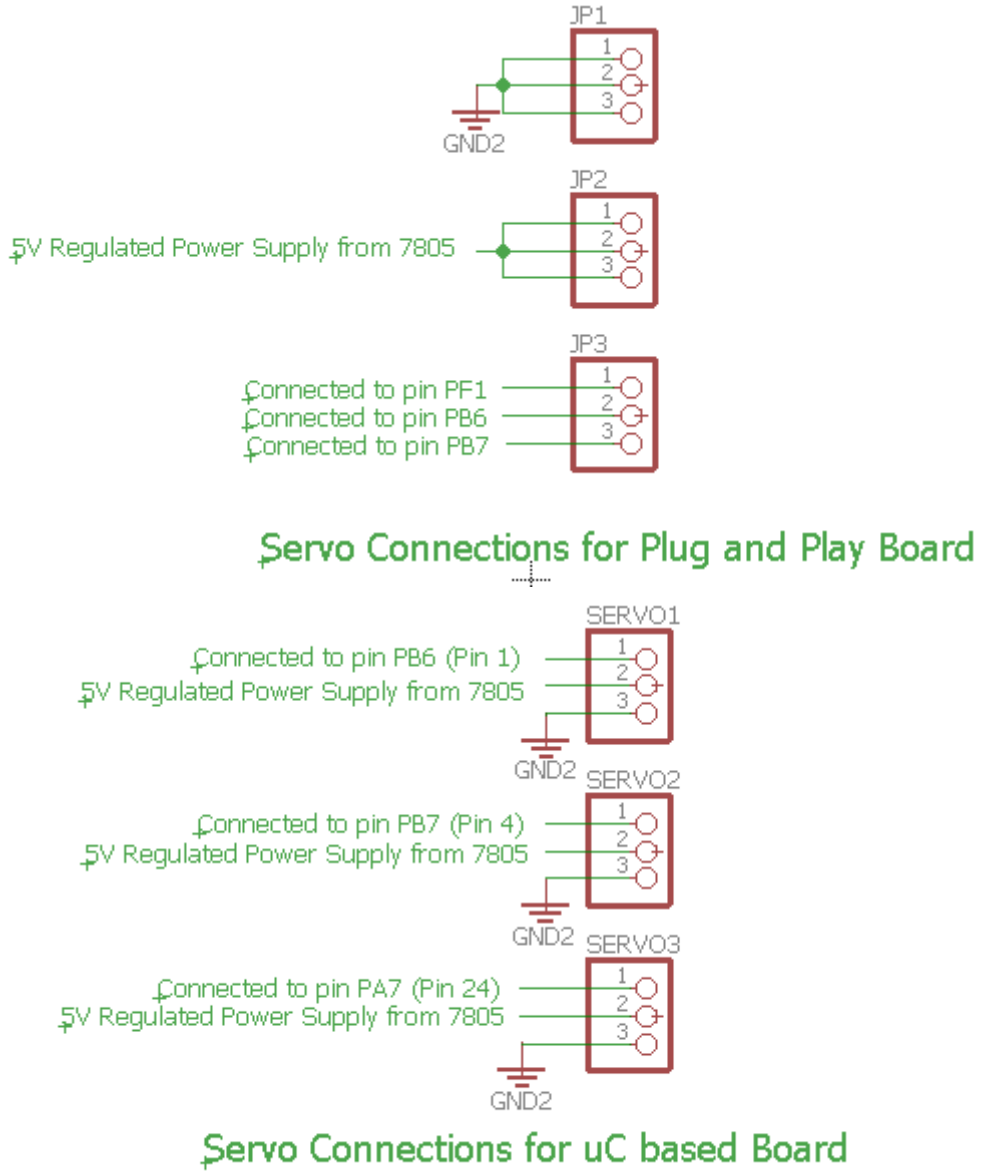
## 5.9 Reset Switch

The Plug and play board makes use of reset button present on the TIVA launchpad. The uC based has a switch connected to the reset the reset pin 38 of the microcontroller. The schematic is given below.




## 5.10 Servo Connectors

The microcontroller board has three Servo connectors. It can be used for driving servo motors of camera pod or any other attachment. Power for the servo connector is provided by the “5V servo supply” voltage regulator. Both the board have different pwm pins for servo which can be seen from the schematic.



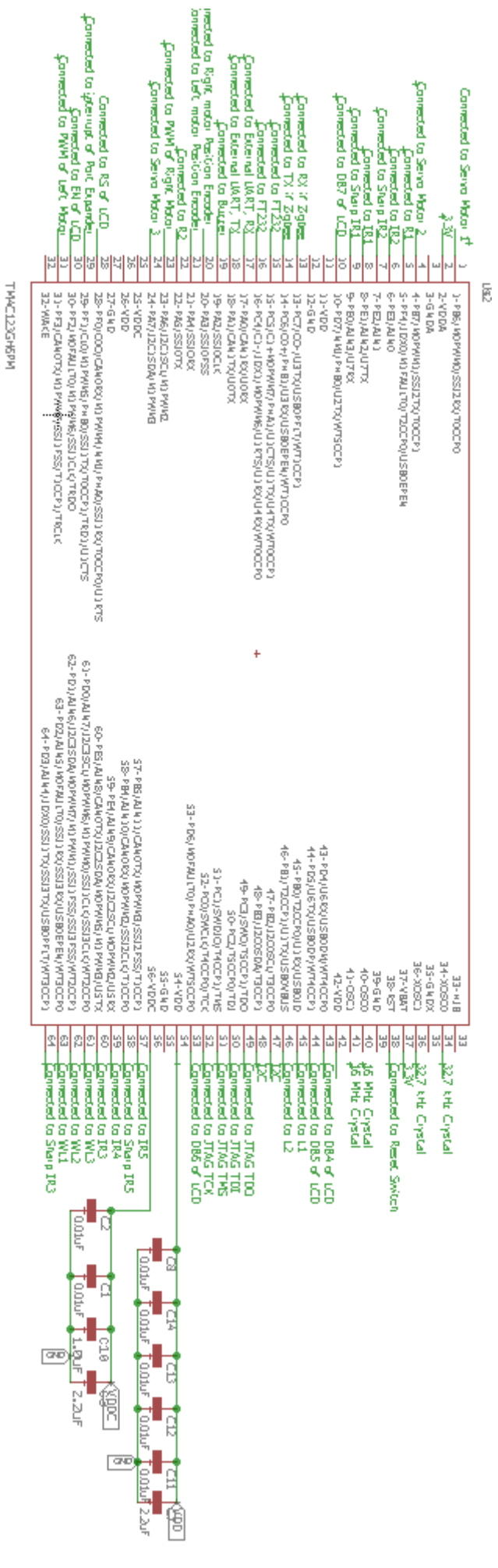

## 5.11 TM4C123GH6PM Micro-controller:

The Tiva™ C Series ARM Cortex-M4 microcontrollers provide top performance and advanced integration. The product family is positioned for cost-conscious applications requiring significant control processing and connectivity capabilities such as:

- Low power, hand-held smart devices
- Gaming equipment
- Home and commercial site monitoring and control
- Motion control
- Medical instrumentation
- Test and measurement equipment
- Factory automation
- Fire and security

- Smart Energy/Smart Grid solutions
- Intelligent lighting control
- Transportation

Schematic Of the connections is shown below.


TM4C123GH6PM
U$2
Connected to Servo Motor 1
Connected to Servo Motor 2
Connected to B1
Connected to IB2
Connected to Sharp IB1
Connected to Sharp IB4
Connected to DB7 of LCD
Connected to RX if Zigbee
Connected to TX if Zigbee
Connected to FT232
Connected to FT232
Connected to External UART TX
Connected to External UART RX
Connected to Buzzer
Connected to Right motor Position Encoder
Connected to Left motor Position Encoder
Connected to B2
Connected to PWM of Right Motor
Connected to Servo Motor 3
Connected to interrupt of Port Expander
Connected to EN of LCD
Connected to PWM of Left Motor
Connected to RS of LCD
1-PB6/M0PWM0/SSI2CLK/TOCCP0
2-3.3V
3-GND
4-PB7/M0PWM1/SSI2TX/TOCCP1
5-PF4/U1DSR/M1FAULT0/T2CCP0/USB0EPEN
6-PE3/AIN0
7-PE2/AIN1
8-PE1/AIN2/U7TX
9-PE0/AIN3/U7RX
10-PD7/NMI/PHB0/U2TX/WT5CCP1
11-VDD
12-GND
13-PC7/CO-/U3TX/USB0PFLT/WT3CCP1
14-PC6/CO+/PHB1/U3RX/USB0EPEN/WT3CCP0
15-PC5/C1+/M0PWM7/PHA1/U1CTS/U1TX/U4TX/WT2CCP1
16-PC4/C1-/U1DSR/M0PWM6/U1RTS/U1RX/U4RX/WT2CCP0
17-PA0/CAN1RX/U0RX
18-PA1/CAN1TX/U0TX
19-PA2/SSI0CLK
20-PA3/SSI0FSS
21-PA4/SSI0RX
22-PA5/SSI0TX
23-PA6/I2C1SCL/M1PWM2
24-PA7/I2C1SDA/M1PWM3
25-VBDC
26-VDD
27-GND
28-PF0/C0O/CAN0RX/M1PWM4/PHA0/SSI1RX/TOCCP0/U1RTS
29-PF1/C1O/M1PWM5/PHB0/SSI1TX/TOCCP1/TRD1/U1CTS
30-PF2/M0FAULT0/M1PWM6/SSI1CLK/TRD0
31-PF3/CAN0TX/M1PWM7/SSI1FSS/T1CCP1/TRCLK
32-WAKE
33-HIB
34-XOSC0
35-GNDX
36-XOSC1
37-VBAT
38-RST
39-GND
40-OSC0
41-OSC1
42-VDD
43-PD4/U6RX/USB0DM/WT4CCP0
44-PD5/U6TX/USB0DP/WT4CCP1
45-PB0/T2CCP0/U1RX/USB0ID
46-PB1/T2CCP1/U1TX/USB0VBUS
47-PE0/I2C0SCL/T3CCP0
48-PE1/I2C0SDA/T3CCP1
49-PC3/SWO/TSCCP1/TDO
50-PC2/TSCCP0/TDI
51-PC1/SWDIO/T4CCP1/TMS
52-PC0/SWCLK/T4CCP0/TCK
53-PD6/M0FAULT0/PHA0/U2RX/WT5CCP0
54-VDD
55-GND
56-VBDC
57-PB5/AIN11/CAN0TX/M0PWM3/SSI2FSS/T1CCP1
58-PB4/AIN10/CAN0RX/M0PWM2/SSI2CLK/T1CCP0
59-PE4/AIN9/CAN0RX/I2C2SCL/M0PWM4/U5RX
60-PB5/AIN8/CAN0TX/I2C2SDA/M0PWM5/M1PWM3/U5TX
61-PD0/AIN7/I2C3SCL/M0PWM6/M1PWM0/SSI1CLK/SSI3CLK/WT2CCP0
62-PD1/AIN6/I2C3SDA/M0PWM7/M1PWM1/SSI1FSS/SSI3FSS/WT2CCP1
63-PD2/AIN5/M0FAULT0/SSI1RX/SSI3RX/USB0EPEN/WT3CCP0
64-PD3/AIN4/U1DSR/SSI1TX/SSI3TX/USB0PFLT/WT3CCP1
Connected to IR5
Connected to Sharp IR5
Connected to IR4
Connected to IR3
Connected to WL3
Connected to WL2
Connected to WL1
Connected to Sharp IR3
C2
0.01uF
C1
0.01uF
C10
1.0uF
2.2uF
VDDC
GND
C9
0.01uF
C14
0.01uF
C13
0.01uF
C12
0.01uF
C11
0.01uF
2.2uF
VDD

# 5.12 Pin Functionality

## 5.12.1 Pin of uC

| Pin No. | Pin | Complete Pin Connections |
|---|---|---|
| 1 | PB6 | Servo Motor 1 |
| 2 | VDDA | VDD filtered through capcitors |
| 3 | GNDA | Ground |
| 4 | PB7 | Servo Motor 2 |
| 5 | PF4 | Pin 53 Right Motor 1 |
| 6 | PE3 | 16 IR proximity Sensor 2 |
| 7 | PE2 | Output of Second OpAmp of lm324 |
| 8 | PE1 | 12 IR Proximity Sensor 1 |
| 9 | PE0 | output of First OpAmp of lm324 |
| 10 | PD7 | 28 DB7 of LCD Data |
| 11 | VDD | VDD filtered through capcitors |
| 12 | GND | Ground |
| 13 | PC7 | 13 Zigbee Rx |
| 14 | PC6 | 14 Zigbee Tx |
| 15 | PC5 | FT232 |
| 16 | PC4 | FT232 |
| 17 | PA0 | External UART |
| 18 | PA1 | External UART |
| 19 | PA2 | Buzzer |
| 20 | PA3 | 63 Right Position Encoder Interrupt |
| 21 | PA4 | 64 Left Position Encoder Interrupt |
| 22 | PA5 | 55 Right Motor 2 |
| 23 | PA6 | 54 Right Motor Pwm |
| 24 | PA7 | Servo Motor 3 |
| 25 | VDDC | Connected to VDDC on 56 |
| 26 | VDD | VDD filtered through capcitors |
| 27 | GND | Ground |
| 28 | PF0 | 22 RS of LCD |
| 29 | PF1 | INT A and B of to GPIO expander shorted and connected |
| 30 | PF2 | 24 EN of LCD |
| 31 | PF3 | 50 Left Motor PWM |
| 32 | WAKE | Ground |
| 33 | HIB | NC |
| 34 | XOSC0 | 32.7 KHz crystals(One End) |
| 35 | GNDX | Cap to crystal |
| 36 | XOSC1 | 32.7 KHz crystals(Other End) |
| 37 | VBAT | 3.3 Volts |
| 38 | RST | Reset Switch |
| 39 | GND | Ground |
| 40 | OSC1 | 16 MHz crystal(One end) |
| 41 | OSC1 | 16 MHz crystal(Other end) |
| 42 | VDD | VDD filtered through capcitors |
| 43 | PD4 | 26 DB4 Of LCD |
| 44 | PD5 | 25 DB5 of LCD |
| 45 | PB0 | 51 Left Motor 1 |

| | | |
|---|---|---|
| 46 | PB1 | 52 Left Motor 2 |
| 47 | PB2 | I2C ADC SCL |
| 48 | PB3 | I2C ADC SDA |
| 49 | PC3 | JTAG TDO |
| 50 | PC2 | JTAG TDI |
| 51 | PC1 | JTAG TMS |
| 52 | PC0 | JTAG TCK |
| 53 | PD6 | 27 DB6 Of LCD |
| 54 | VDD | VDD filtered through capcitors |
| 55 | GND | Ground |
| 56 | VDDC | Connected to VDDC on 25 |
| 57 | PB5 | 46 IR Proximity Sensor 5 |
| 58 | PB4 | Output of First OpAmp of lm358 |
| 59 | PE4 | 43 IR Proximity Sensor 4 |
| 60 | PE5 | 42 IR Proximity Sensor 3 |
| 61 | PD0 | 32 White Line Sensor 3 |
| 62 | PD1 | 31 White Line Sensor 2 |
| 63 | PD2 | 30 White Line Sensor 1 |
| 64 | PD3 | Output of Third OpAmp of lm324 |

### 5.12.2 Robot Main Board Connections

| Pin Out | Pin Name | Functionality | PIN on DB | Pin on Pluggable B |
|---|---|---|---|---|
| 1 | Current sensor | Current sense analog value | Not Using | |
| 2 | IR Proximity sensor 8 | IR Proximity sensor 8 analog value | External Adc INT0 | |
| 3 | GND | Ground | Ground | |
| 4 | DATA+ | USB connection going to the ATMEGA2560 USB connection with uC | C4 | |
| 5 | DATA- | microcontroller via FT232 USB to serial USB connection with uC | PC5 | |
| 6 | VCC | USB converter. Connect TO VCC of FT232 | | |
| 7 | 5V System | "5V System Voltage. Can be used for powering up any digital device with current limit of 400mA." | | |
| 8 | 5V System | "5V System Voltage. Can be used for powering up any digital device with current limit of 300mA." | | |
| 9 | 5V System | "5V System Voltage. Can be used for powering up any digital device with current limit of 300mA." | | |
| 10 | 5V System | "5V System Voltage. Can be used for powering up any digital device with current limit of 400mA." | | |
| 11 | SHARP IR Range Sensor 1 | Analog output of Sharp IR range Sensor 1 | PE0(lm3241) | |
| 12 | IR Proximity Sensor 1 | Analog output of IR Proximity sensor 1 | PE1 | |

| | | | | |
|---|---|---|---|---|
| 13 | XBee RXD | XBee wireless module Serial data in | PC7 | |
| 14 | XBee TXD | XBee wireless module Serial data out | PC6 | |
| 15 | SHARP IR Range Sensor 2 | Analog output of Sharp IR range sensor 2 | PE2(lm3242) | |
| 16 | IR Proximity Sensor 2 | Analog output of IR Proximity sensor 2 | PE3 | |
| 17 | RSSI | To capture the RSSI signal | | |
| 18 | MOSI | MOSI of the Controller/NC create extra expansion headers | | |
| 19 | MISO | MISO of controller/NC create extra expansion headers | | |
| 20 | SCK | SCK of the controller/NC create extra expansion headers | | |
| 21 | SSI | SS of the controller/ NC create extra expansion headers | | |
| 22 | RS | connected to RS of LCD normal I/O | PF0 | |
| 23 | RW | connected to RW of LCD normal I/O | GND | |
| 24 | EN | connected to EN of LCD normal I/O | PF2 | |
| 25 | DB5 | data pin of lcd normal I/O | PD5 | |
| 26 | DB4 | data pin of lcd normal I/O | PD4 | |
| 27 | DB6 | data pin of lcd normal I/O | PD6 | |
| 28 | DB7 | data pin of lcd normal I/O | PD7 | |
| 29 | V Battery System | ADC to check the level of battery voltage | | |
| 30 | WL1 | Analog output of white line sensor 1 | PD2 | |
| 31 | WL2 | Analog output of white line sensor 2 | PD1 | |
| 32 | WL3 | Analog output of white line sensor 3 | PD0 | |
| 33 | ”Sharp IR Sensors 1and 5 Disable” | | | |
| 34 | IR Proximity Sensor Disable | | | |
| 35 | 5V System | ”5V system Voltage. Can be used for powering | up any digital device. Current Limit: 400mA.” | |
| 36 | WL4 | Analog output of white line sensor 4 | External Adc INT1 | |
| 37 | WL5 | Analog output of white line sensor 5 | External Adc INT2 | |
| 38 | WL6 | Analog output of white line sensor 6 | External Adc INT3 | |
| 39 | WL7 | Analog output of white line sensor 7 | External Adc INT4 | |
| 40 | White Line Sensors Disable | | | |
| 41 | Sharp IR Range Finder 3 | Analog output of Sharp IR range sensor 3 | PD3 (lm 324 3) | |
| 42 | IR Proximity Sensor 3 | Analog output of IR Proximity sensor 3 | PE5 | |
| 43 | IR Proximity Sensor 4 | Analog output of IR Proximity sensor 4 | PE4 | |

| | | | | |
|---|---|---|---|---|
| 44 | Sharp IR Range Finder 4 | Analog output of Sharp IR range sensor 4 | in 5 ex (lm 324 5) | |
| 45 | Sharp IR Range Finder 5 | Analog output of Sharp IR range sensor 5 | PB4 (lm358 1) | |
| 46 | IR Proximity Sensor 5 | Analog output of IR Proximity sensor 5 | PB5 | |
| 47 | C11 | motor not present | | |
| 48 | C1 | PWM not present | | |
| 49 | c12 | not present | | |
| 50 | PWM L | left motor PWM(timer pin in PWM mode) | PF3 | |
| 51 | L1 | left motor pin1 normal I/O | PB0 | |
| 52 | L2 | left motor pin2 normal I/O | PB1 | |
| 53 | R1 | right motor pin1 normal I/O | PF4 | |
| 54 | PWM R2 | right motor PWM(timer pin in PWM mode) | PA6 | |
| 55 | R2 | right motor pin2 | PA5 | |
| 56 | NC | | | |
| 57 | NC | | | |
| 58 | NC | | | |
| 59 | NC | | | |
| 60 | NC | | | |
| 61 | NC | | | |
| 62 | Position encoder left | Output of Left position encoder (0-5V) PA4 | | |
| 63 | Position encoder right | Output of Right position encoder (0-5V) PA3 | | |
| 64 | position enocder C2 | Output of C2 position encoder (0-5V) | | |
| 65 | Position encoder C1 | Output of C1 position encoder (0-5V) | | |
| 66 | C22 | NC | | |
| 67 | C21 | NC | | |
| 68 | C2 | Pwm | NC | |
| 69 | IR Prox6 | Analog output of IR Proximity sensor 6 External Adc | INT6 | |
| 70 | IR Prox7 | Analog output of IR Proximity sensor 7 External Adc | INT7 | |
| 71 | Buzzer | Input, V¿0.65V turns on the Buzzer | PA2 | |
| 72 | DAC Out | NC | | |
| 73 | RS232 TX | NC | | |
| 74 | RS232 RX | NC | | |

#### 5.12.3 Pin Connection Of Plug And Play Board

| Pin Name | Pin Connection on Main Board | Function |
|---|---|---|
| PA0 | | Used for Programming |
| PA1 | | Used for Programming |
| PA2 | 26 | DB4 of LCD |
| PA3 | 27 | DB5 of LCD |
| PA4 | 28 | DB6 of LCD |
| PA5 | 29 | DB7 of LCD |
| PA6 | | I2C |

| | | |
|---|---|---|
| PA7 | | I2C |
| PB0 | 14 | Zigbee Tx |
| PB1 | 13 | Zigbee RX |
| PB2 | 62 | Position encoder of left motor |
| PB3 | 52 | L2 |
| PB4 | 11 | Sharp IR1 |
| PB5 | 12 | IR 1 |
| PB6 | | Servo |
| PC0 | | |
| PC1 | | |
| PC2 | | |
| PC3 | | |
| PC4 | 53 | R1 |
| PC5 | 54 | PWM of right motor |
| PC6 | 55 | R2 |
| PC7 | | Interrupt of port expander |
| PD0 | 16 | IR 2 |
| PD1 | 43 | IR Prox 4 |
| PD2 | 41 | Sharp IR 3 |
| PD3 | 42 | IR Prox 3 |
| PD4 | | |
| PD5 | | |
| PD6 | 22 | RS of LCD |
| PD7 | 24 | EN of LCD |
| PE0 | 70 | IR 7 |
| PE1 | 30 | WL1 |
| PE2 | 31 | WL2 |
| PE3 | 32 | WL3 |
| PE4 | 45 | Sharp IR 5 |
| PE5 | 46 | IR 5 |
| PE6 | | |
| PE7 | | |
| PF0 | 63 | Position encoder of right motor |
| PF1 | | Servo |
| PF2 | 50 | PWM of left motor |
| PF3 | 51 | L1 |
| PF4 | 71 | Buzzer |

# 6 Software Manual:

## 6.1 Code Composer Studio:

Code Composer Studio is an integrated development environment (IDE) that supports TI's Microcontroller and Embedded Processors portfolio. Code Composer Studio comprises a suite of tools used to develop and debug embedded applications. It includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features. The intuitive IDE provides a single user interface taking you through each step of the application development flow. Familiar tools and interfaces allow users to get started faster than ever before. Code Composer Studio combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature-rich development environment for embedded developers. This description is directly taken from the website of Texas Instruments and click to know more about CC Studio

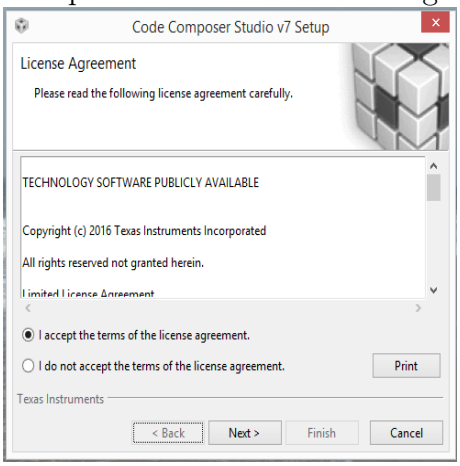
### 6.1.1 Download CC Studio:

At the time of writing this document Version 7 was the latest one. You can check for the latest at Download CCS.(do not download any beta versions).There will be two installer files.The web installer will require Internet access until it completes. If the web installer version is unavailable or you can't get it to work, download, unzip and run the offline version. The offline download will be much larger than the installed size of CCS since it includes all the possible supported hardware.

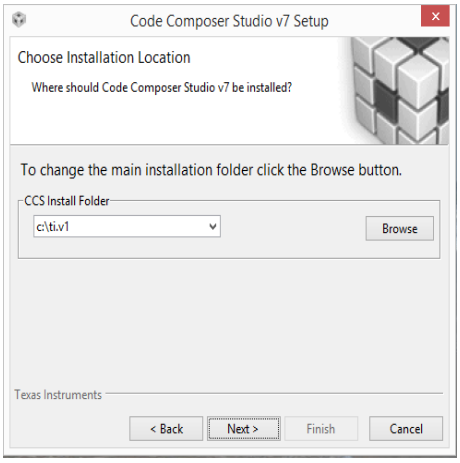
### 6.1.2 Installing C C Studio:

After the installer has started follow the steps mentioned below:

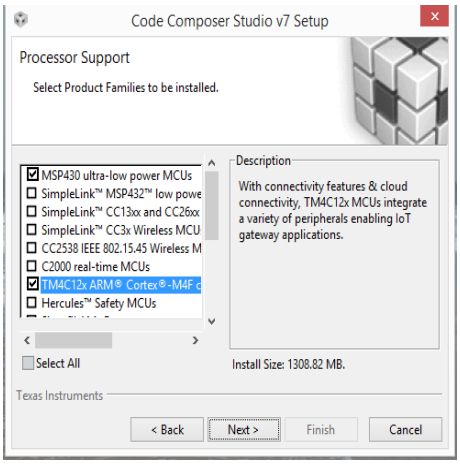
1. Accept the Software License Agreement and click Next.



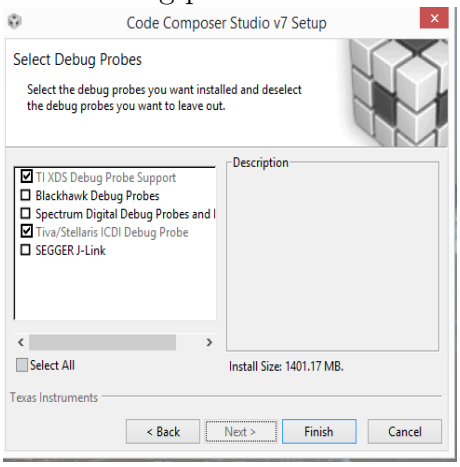

2. Select the destination folder and click next.




3. Select the processors that your CCS installation will support. You must select "TM4C12X Arm Cortex M4". You can select other architectures, but the installation time and size will increase.




4. Select debug probes and click finish




5. The installer process should take 15 - 30 minutes, depending on the speed of your connection. The offline installation should take 10 to 15 minutes. When the installation is complete, uncheck the "Launch Code Composer Studio v7" checkbox and then click Finish.There are several additional tools that require installation during the CCS install process. Click "Yes" or "OK" to proceed when these appear.

6. Install TivaWare for C Series (Complete). Download and install the latest full version of TivaWare from: TivaWare. The filename is SW-TM4C-x.x.exe . This workshop was built using version 1.1. Your version may be a later one. If at all possible, please install TivaWare into the default location.

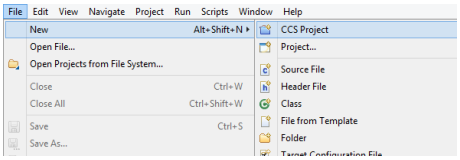
**You can find additional information at these websites:**
Main page: www.ti.com/launchpad
Tiva C Series TM4C123G LaunchPad:
http://www.ti.com/tool/ek-tm4c123gxl
TM4C123GH6PM folder:
http://www.ti.com/product/tm4c123gh6pm
BoosterPack webpage: www.ti.com/boosterpack
LaunchPad Wiki:
www.ti.com/launchpadwiki

For understanding the launchpad properly and to learn more about Tiva it is strongly recommended to go through the webpage TIva Worshops and download and read the workbook

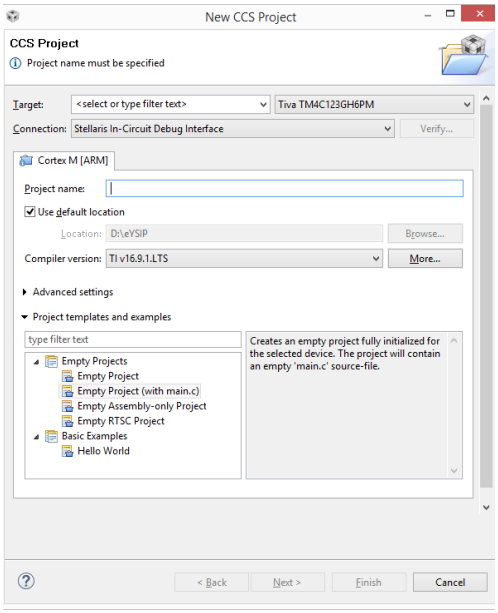
### 6.1.3 Create a New Project

To create new project follow the steps mentioned:

1. Click File then New and then CCS Projects




2. Select Target and connection as shown in the photo. Give a name to your project and save in a location.Click Finish. A main.c file will be open




### 6.1.4 Add Path and Build Variables

The path and build variables are used for:

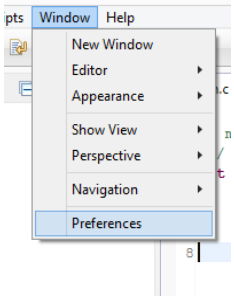
- Path variable – when you ADD (link) a file to your project, you can specify a "relative to" path. The default is PROJECT_LOC which means that your linked resource (like a .lib file) will be linked relative to your project directory.
- Build variable – used for items such as the search path for include files associated with a library – i.e. it is used when you build your project.

Variables can either have a PROJECT scope (that they only work for this project) or a WORKSPACE scope (that they work across all projects in the workspace). In the next step, we need to add (link) a library file and then add a search path for include files. First, we'll add these variables MANUALLY as WORKSPACE variables so that any project in your workspace can use the variables. Refer to the workbook by TI for adding as PROJECT

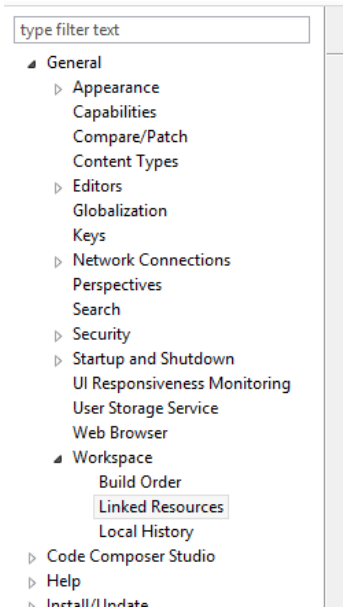
#### 6.1.4.1 Adding a Path Variable

To add a path variable,:

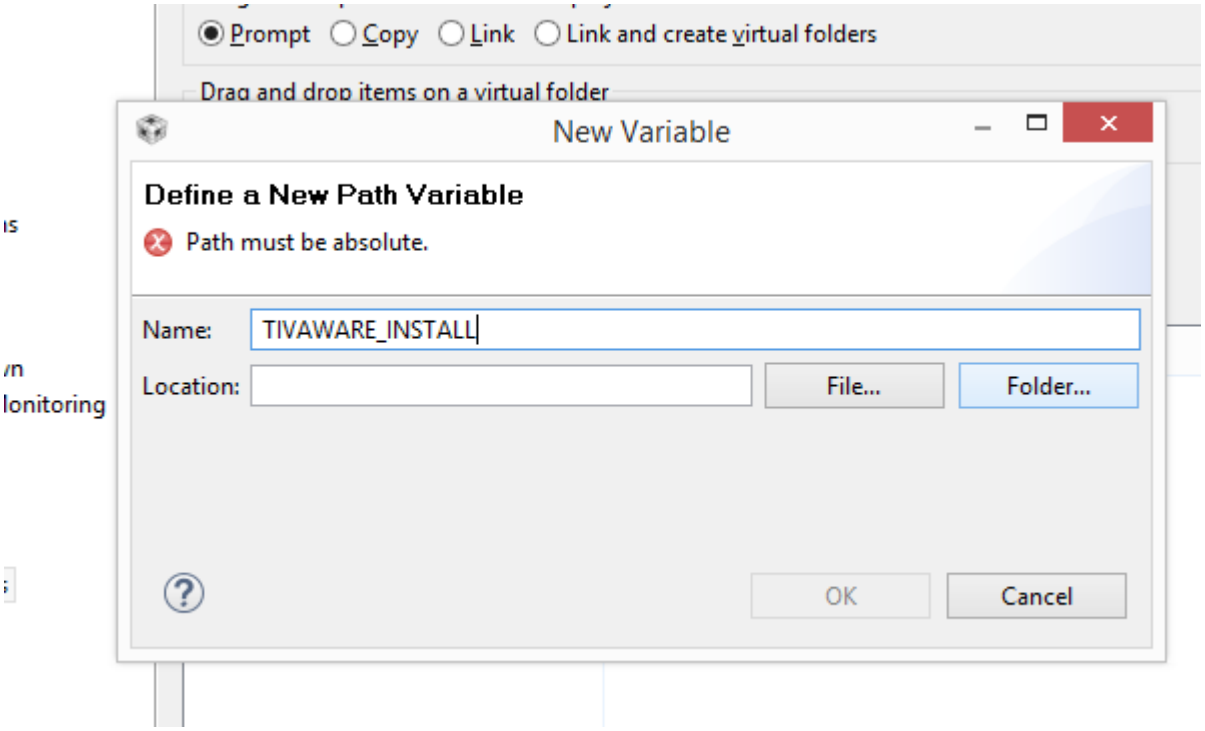
- Right-click on your Window Tab and select Preference.



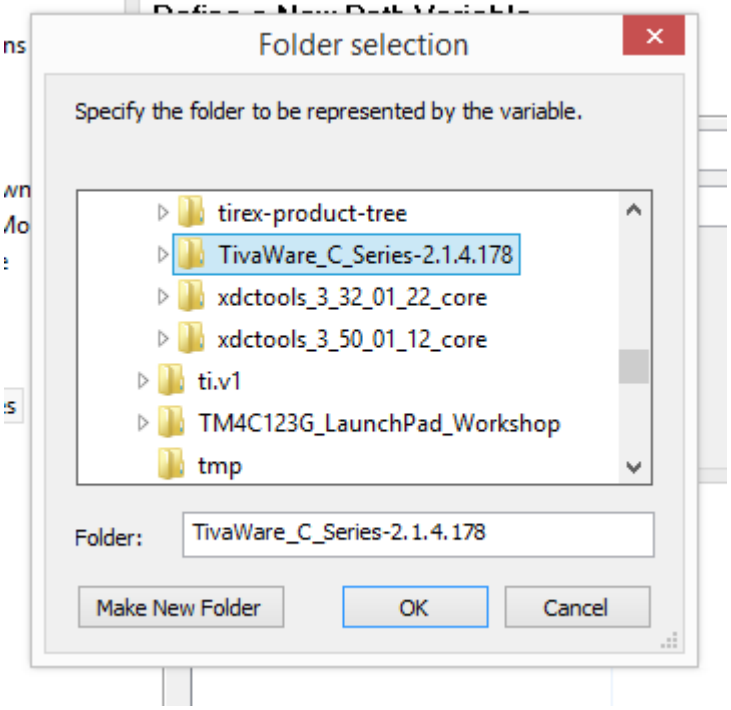

- Expand General list in the upper left-hand corner as shown and then expand the Resource list and click on Linked Resources: We want to add a New variable to specify exactly where you installed TivaWare.

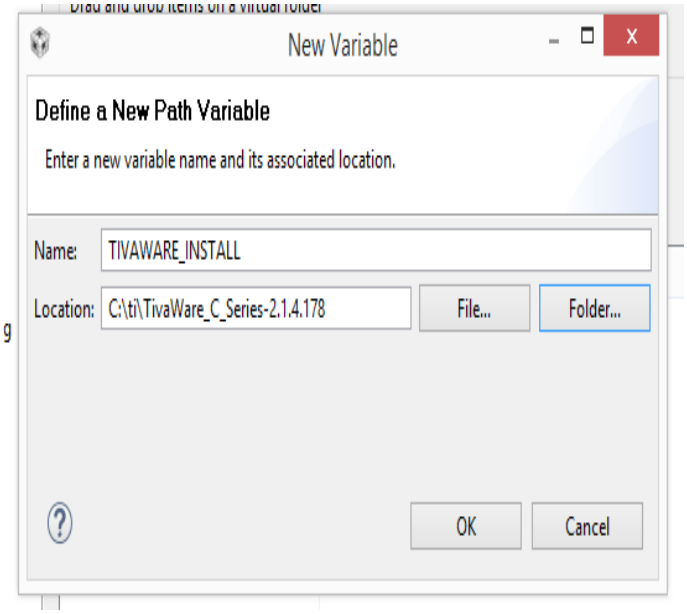



- Click New
- When the New Variable dialog appears, type TIVAWARE_INSTALL for the name.

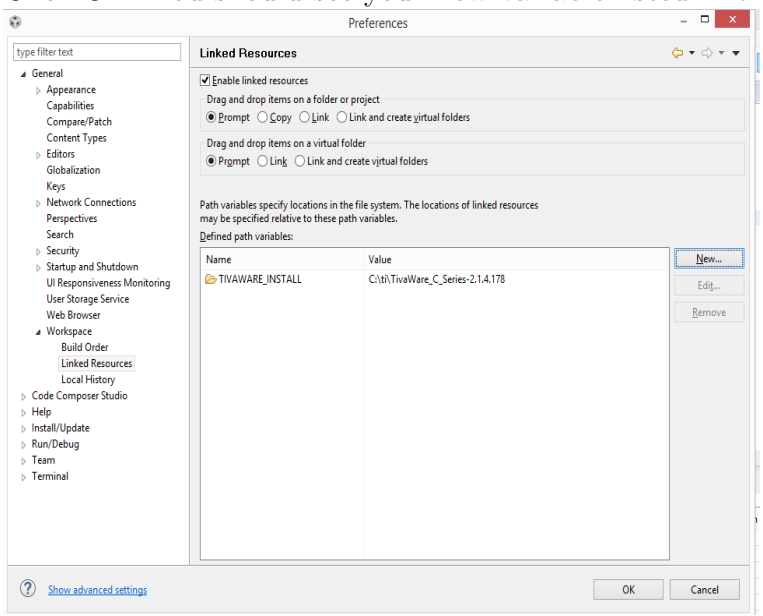



- For the Location, click the Folder... button and navigate to your TivaWare installation. Click on the folder name and then click OK.





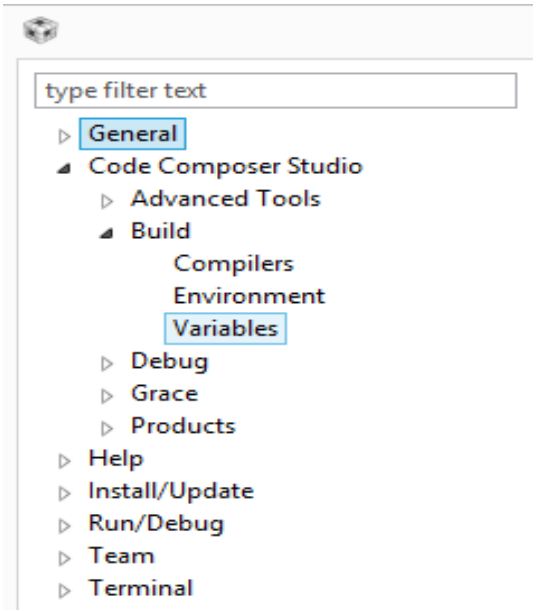

- Click OK. You should see your new variable listed in the Variables list.




### 6.1.4.2 Adding a Build Variable

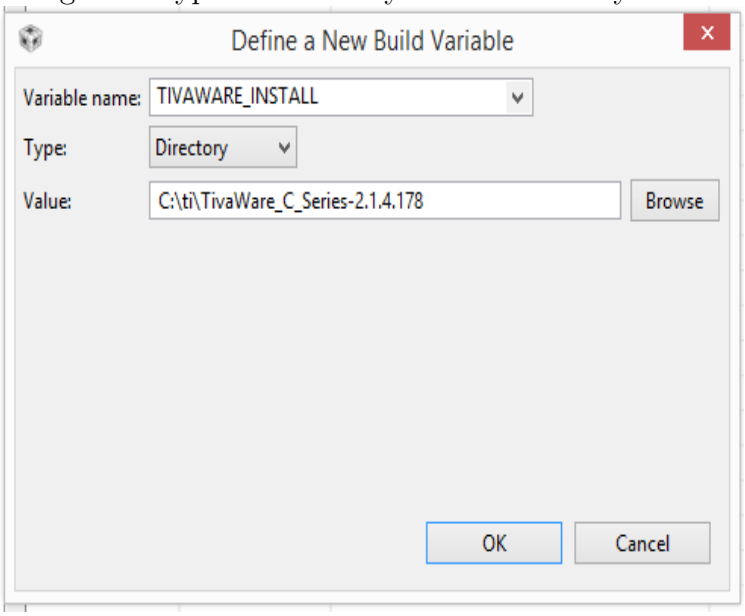
Now let's add a build variable that we will use in the include search path for the INCLUDE files associated with the TivaWare driver libraries.

- Click on Code Composer Studio Build and then the Variables tab:

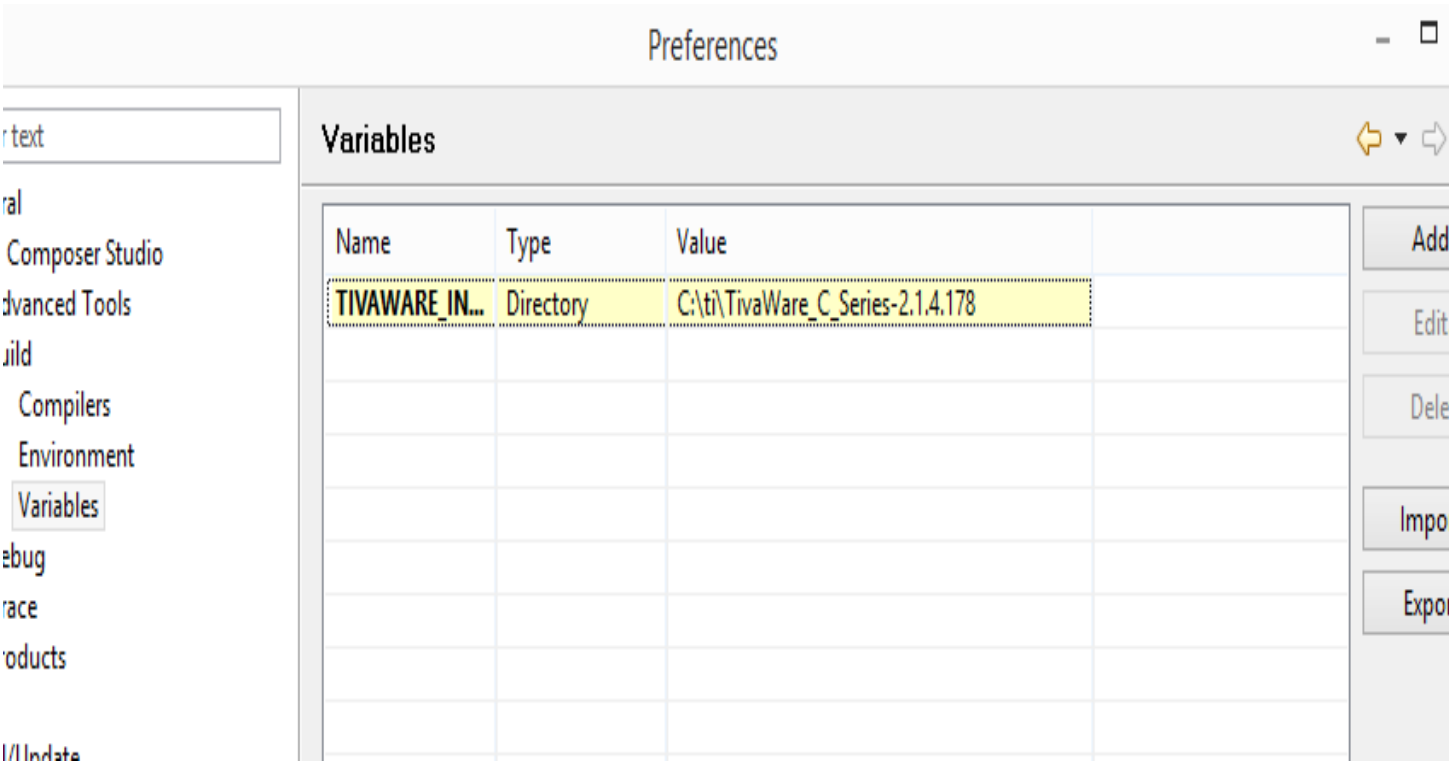



- Click the Add button. When the Define a New Build Variable dialog appears, insert TIVAWARE_INSTALL into the Variables name box.

- Change the Type to Directory and browse to your Tivaware installation folder.




- Click OK.
- Click OK again to save and close the Build Properties window.




## 6.2 driver.lib

## 6.3 Buzzer

Located in the folder "Buzzer_Beep" folder in the documentation. In this example, we will load buzzer beep code in Tiva based Fire Bird V. Now we will see in detail the structure of this code.This experiment demonstrates the simple operation of Buzzer ON/OFF with one second delay. Buzzer is connected to PORTF 4 pin of the Tiva Launchpad. If you have uC based board then it is connected to PORTA 2.

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
//This header File is important to Unlock GPIO Pins
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

/**** Useful Macros Definition******/
/******Remove the comments if you are using uC board******
#define buzzerEnable    SYSCTL_PERIPH_GPIOA
#define buzzer          GPIO_PORTA_BASE
#define buzzerPin       GPIO_PIN_2
*********************************************************/

/******Remove the comments if you are using uC board******/
#define buzzerEnable    SYSCTL_PERIPH_GPIOF
#define buzzer          GPIO_PORTF_BASE
#define buzzerPin       GPIO_PIN_4
/*********************************************************/

#define buzzerOn()      GPIOPinWrite(buzzer,buzzerPin,255)
#define buzzerOff()     GPIOPinWrite(buzzer,buzzerPin,0)
/*********************************/
void setupCLK();
void peripheralEnable();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_ms(uint64_t delay);

int main(void) {
```

```
33          setupCLK();
34          peripheralEnable();
35          configIOPin();
36          while(1){
37            buzzerOn();
38            delay_ms(1000);
39            buzzerOff();
40            delay_ms(1000);
41          }
42        }
43        /*****************************************************************************
44        * This function is used to setup Clock frequency of the controller
45        * It can be changed through codes
46        * In this we have set frequency as 40Mhz
47        * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
48        *****************************************************************************/
49        void setupCLK(){
50          SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
51        }
52        /*****************************
53        * Enabling System Peripherals
54        * buzzer Port in this case
55        * buzzerPin for buzzer output
56        *****************************/
57        void peripheralEnable(){
58          SysCtlPeripheralEnable(buzzerEnable);
59          /***** Just in case you are not familiar with macros*****
60          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
61          ************This is enabling PORTF*********************/
62        }
63        /***********************************
64        * Configuring Pin as Input Or Output
65        ***********************************/
66        void configIOPin(){
67          GPIOPinTypeGPIOOutput(buzzer, buzzerPin);
68          /***** Just in case you are not familiar with macros*****
69          GPIOPinTypeGPIOOutput(buzzer, GPIO_PIN_4);
70          *****************This is P4 output*********************/
71        }
72        /*****************************************
73        * Calculating Delays
74        * extern void SysCtlDelay(uint32_t ui32Count)
75        * waits until the counting has been completed
76        *****************************************/
77        void delay_ms(uint64_t delay){
78          SysCtlDelay(delay*SysCtlClockGet()/3000.0);
79        }
80        void delay_us(uint64_t delay){
81          SysCtlDelay(delay*SysCtlClockGet()/3000000.0);
82        }
83
```

## 6.4 Programming the Robot

## 6.5 Using Debugger of The Programmer

## 6.6 Simple I/O Operation

This experiment demonstrates the simple I/O operations. This example is only for plug and play board, but this should not discourage the user from understanding the example as it provide very important example of I/O operation on TIVA platform.
In this lab you have to use switch SW1, SW2 and RGB LED present on Tiva C series board.
1. Use switch SW1 to Turn on Red LED on first switch press, Green LED on second switch press and Blue LED on third switch press. Repeat the same cycle next switch press onwards. Note that LED should remain on for the duration switch is kept pressed i.e. LED should turn off when switch is released.
2. Use switch SW2 and sw2Status (a variable). Your program should increment sw2Status by one, every time switch is pressed. Note how the value of sw2Status changes on each switch press. Use debugger and

add sw2Status to Watch Expression" window. (You will find Continuous Refresh button on top of the Expression Window). You can use step debugging or breakpoints to check the variable value. Hint:To add variable to Expression Window, select and right click the variable name and select Add Watch Expression". To view Expression Window, click on View button from CCS menu bar and select Expressions.

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h" //To unlock locked pins for GPIO
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#define userSwitch1 GPIO_PIN_0
#define redLed      GPIO_PIN_1
#define blueLed     GPIO_PIN_2
#define greenLed    GPIO_PIN_3
#define userSwitch2 GPIO_PIN_4
#define LOCK_F (*((volatile unsigned long *)0x40025520))
#define CR_F   (*((volatile unsigned long *)0x40025524))
void setupCLK();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_us(uint64_t delay);

int main(){
  setupCLK();
  SysCtlDelay(3);
  configIOPin();
  unsigned char pinData=1;
  unsigned char state=2;
  unsigned char countSwitch2=0;
  unsigned char flagSW1=0;
  unsigned char flagSW2=0;
  while(1){
    pinData=GPIOPinRead(GPIO_PORTF_BASE,userSwitch2|userSwitch1);
    if((pinData&0x01)==0)
      flagSW1=1;
    else if((flagSW1==1)&&(pinData&0x01)==1){
      countSwitch2+=1;
      flagSW1=0;
    }
    if((pinData&0x10)==0){
      GPIOPinWrite(GPIO_PORTF_BASE,redLed|blueLed|greenLed,state);
      flagSW2=1;
    }
    else if(((flagSW2==1)&&(pinData&0x10)==0x10)){
      flagSW2=0;
      GPIOPinWrite(GPIO_PORTF_BASE,redLed|blueLed|greenLed,0);
      state=state*2;
      if(state>8)
        state=2;
    }
    delay_ms(5);
  }
}
/*******************************************************************************
* This function is used to setup Clock frequency of the controller
* Enabling System Peripherals
* PORTF in this case
*****************************/
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
}
/*******************************************
* Configuring Pin as Input Or Output
* PF0 by default is locked and cannot
* be used as input unless it is unlocked
*******************************************/
void configIOPin(){
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, redLed|blueLed|greenLed);
  HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
  HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
  HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
  GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, userSwitch2|userSwitch1);
  GPIOPadConfigSet(GPIO_PORTF_BASE ,userSwitch2|userSwitch1,GPIO_STRENGTH_12MA,
```

```
    GPIO_PIN_TYPE_STD_WPU);
72      }
73      /**********************************
74      * Calculating Delays
75      **********************************/
76      void delay_ms(uint64_t delay){
77        SysCtlDelay(delay*(SysCtlClockGet()/3000));
78      }
79      void delay_us(uint64_t delay){
80        SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
81      }
82
```

# 6.7 Robot Direction Control

Located in the folder "Experiments  Motion Control Simple" folder. Robot's motors are controlled by L293D motor controller. Using L293D, microcontroller can control direction and velocity of both of the motors. To change the direction appropriate logic levels (High/Low) are applied to IC L293D's direction pins. Velocity control is done using pulse width modulation (PWM) applied to Enable pins of L293D IC.

The Motor connections are as shown below

| Motor Pin | Pin Name(uC) | Pin Name(Plug and Play) |
|---|---|---|
| L1 | PB0 | PF3 |
| L2 | PB1 | PB3 |
| PWM L | PF3 | PF2 |
| R1 | PF4 | PC4 |
| R1 | PA5 | PC6 |
| PWM R | PA6 | PC5 |

### Code for Plug and Play Board:

```
1       #include <stdint.h>
2       #include <stdbool.h>
3       #include "inc/hw_types.h"
4       #include "inc/hw_memmap.h"
5       //This header File is important to Unlock GPIO Pins
6       #include "inc/hw_gpio.h"
7       #include "driverlib/sysctl.h"
8       #include "driverlib/gpio.h"
9       //Used for controlling Motor direction
10      #define right           0x41
11      #define left            0x18
12      #define softRight       0x10
13      #define softLeft        0x01
14      #define forward         0x11
15      #define backward        0x48
16
17      void setupCLK();
18      void peripheralEnable();
19      void configIOPin();
20      void delay_ms(uint64_t delay);
21      void delay_us(uint64_t delay);
22      void motion(uint8_t);
23
24      int main(void) {
25        setupCLK();
26        peripheralEnable();
27        configIOPin();
28        while(1){
29          motion(forward);
30          delay_ms(1000);
31          motion(right);
32          delay_ms(1000);
33          motion(left);
34          delay_ms(1000);
35          motion(backward);
36          delay_ms(1000);
37        }
38      }
39      /**********************************************************************************
40      * This function is used to setup Clock frequency of the controller
```

```
        * It can be changed through codes
        * In this we have set frequency as 40Mhz
        * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
        ********************************************************************************/
        void setupCLK(){
          SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
        }
        /*****************************
        * Enabling System Peripherals
        * PORTF,PORTB and PORTC in this case
        *****************************/
        void peripheralEnable(){
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
        }
        /***********************************
        * Configuring Pin as Input Or Output
        * And Setting PWM Pin to Always High
        ***********************************/
        void configIOPin(){
          GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_3);
          GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2);
          GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2,255);
          GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_5,255);
        }
        /***********************************
        * Calculating Delays
        ***********************************/
        void delay_ms(uint64_t delay){
          SysCtlDelay(delay*(SysCtlClockGet()/3000));
        }
        void delay_us(uint64_t delay){
          SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
        }
        /*****************************************************
        * This function is for giving the direction of motion
        * Macros have been defined at starting
        * Macros for directions are 8 bits
        * Out of these 8 bits only 4 are used
        * Bit 0 (LSB) corresponds to PB3
        * Bit 3       corresponds to PF3
        * Bit 4       corresponds to PC4
        * Bit 6       corresponds to PF6
        *****************************************************/
        void motion(uint8_t direction){
          GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_3,direction<<3);
          GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_6,direction);
          GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,direction);
        }

```

## Code for uC based Board:

```
        #include <stdint.h>
        #include <stdbool.h>
        #include "inc/hw_types.h"
        #include "inc/hw_memmap.h"
        //This header File is important to Unlock GPIO Pins
        #include "inc/hw_gpio.h"
        #include "driverlib/sysctl.h"
        #include "driverlib/gpio.h"
        //Used for control Motor direction
        #define right           0x22
        #define left            0x11
        #define softRight       0x02
        #define softLeft        0x10
        #define forward         0x12
        #define backward        0x21
        #define stop            0x00

        void setupCLK();
        void peripheralEnable();
        void configIOPin();
        void delay_ms(uint64_t delay);
        void delay_ms(uint64_t delay);
```

```
23        void motion(uint8_t);
24
25        int main(void) {
26          setupCLK();
27          peripheralEnable();
28          configIOPin();
29          while(1){
30            motion(forward);
31            delay_ms(1000);
32            motion(right);
33            delay_ms(1000);
34            motion(left);
35            delay_ms(1000);
36            motion(backward);
37            delay_ms(1000);
38          }
39        }
40        /*****************************************************************************
41        * This function is used to setup Clock frequency of the controller
42        * It can be changed through codes
43        * In this we have set frequency as 40Mhz
44        * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
45        *****************************************************************************/
46        void setupCLK(){
47          SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
48        }
49        /*****************************
50        * Enabling System Peripherals
51        * PORTF,PORTB and PORTC in this case
52        *****************************/
53        void peripheralEnable(){
54          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
55          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
56          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
57        }
58        /***********************************
59        * Configuring Pin as Input Or Output
60        * And Setting PWM Pin to Always High
61        ***********************************/
62        void configIOPin(){
63          GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1);
64          GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,GPIO_PIN_6|GPIO_PIN_5);
65          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_4|GPIO_PIN_3);
66          GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,255);
67          GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_6,255);
68        }
69        /***********************************
70        * Calculating Delays
71        ***********************************/
72        void delay_ms(uint64_t delay){
73          SysCtlDelay(delay*(SysCtlClockGet()/3000));
74        }
75        void delay_us(uint64_t delay){
76          SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
77        }
78        /*****************************************************
79        * This function is for giving the direction of motion
80        * Macros have been defined at starting
81        * Macros for directions are 8 bits
82        * Out of these 8 bits only 4 are used
83        * Bit 0 (LSB) corresponds to PB0
84        * Bit 1       corresponds to PF1
85        * Bit 4       corresponds to PF4
86        * Bit 5       corresponds to PA5
87        *****************************************************/
88        void motion(uint8_t direction){
89          GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1,direction);
90          GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_5,direction);
91          GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_4,direction);
92        }
93
```

# 6.8 Robot Position Control Using Interrupts

Position encoders give position / velocity feedback to the robot. It is used in closed loop to control robot's position and velocity. Position encoder consists of optical encoder and slotted disc assembly. When this slotted disc moves in between the optical encoder we get square wave signal whose pulse count indicates position and time period indicates velocity.

**Connections:**

**Plug and Play Board:**
PB2 : External interrupt for left motor position encoder
PF0 : External interrupt for the right position encoder
**uC Based Board:**
PA4 : External interrupt for left motor position encoder
PA3 : External interrupt for the right position encoder

### 6.8.1 Calculation of position encoder resolution:

To be added later

**Code for Plug and Play Board:**

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
//This header File is important to Unlock GPIO Pins
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

#define right           0x41
#define left            0x18
#define softRight       0x10
#define softLeft        0x01
#define forward         0x11
#define backward        0x48
#define stop            0x00

void setupCLK();
void peripheralEnable();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_us(uint64_t delay);
void motion(uint8_t);
void interruptEnable();
void encoderInterruptEncountered();
void encoderInterruptEncountered1();
void angleRotate(uint16_t Degrees);
void linearDistanceMM(unsigned int DistanceInMM);
void rightDegrees(unsigned int Degrees);
void leftDegrees(unsigned int Degrees);
void forwardMM(unsigned int DistanceInMM);
void backwardMM(unsigned int DistanceInMM);
volatile unsigned long int ShaftCountRight = 0;
volatile unsigned long int ShaftCountLeft = 0;

int main(void) {
  setupCLK();
  peripheralEnable();
  configIOPin();
  interruptEnable();
  while(1){
    forwardMM(100);
```

```
48            delay_ms(1000);
49            leftDegrees(90);
50            delay_ms(1000);
51          }
52        }
53        /*****************************************************************************
54        * This function is used to setup Clock frequency of the controller
55        * It can be changed through codes
56        * In this we have set frequency as 40Mhz
57        * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
58        *****************************************************************************/
59        void setupCLK(){
60          SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
61        }
62        /*******************************
63        * Enabling System Peripherals
64        * PORTF,PORTB and PORTC in this case
65        *******************************/
66        void peripheralEnable(){
67          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
68          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
69          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
70        }
71        /***********************************
72        * Configuring Pin as Input Or Output
73        * Unlocking PF0
74        * Setting PWM Pins to Always High
75        * Weak Pull to the Input Pins
76        ***********************************/
77        void configIOPin(){
78          GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_3);
79          GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
80          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);
81          GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2,255);
82          GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_5,255);
83          HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
84          HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
85          HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
86          GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);
87          GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU);
88          GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PIN_2);
89          GPIOPadConfigSet(GPIO_PORTB_BASE,GPIO_PIN_2,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU);
90        }
91        /***********************************
92        * Calculating Delays
93        ***********************************/
94        void delay_ms(uint64_t delay){
95          SysCtlDelay(delay*(SysCtlClockGet()/3000));
96        }
97        void delay_us(uint64_t delay){
98          SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
99        }
100       /*************************************************
101       * This function is for giving the direction of motion
102       * Macros have been defined at starting
103       * Macros for directions are 8 bits
104       * Out of these 8 bits only 4 are used
105       * Bit 0 (LSB) corresponds to PB3
106       * Bit 3       corresponds to PF3
107       * Bit 4       corresponds to PC4
108       * Bit 6       corresponds to PF6
109       *************************************************/
110       void motion(uint8_t direction){
111         GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_3,direction<<3);
112         GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_6,direction);
113         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,direction);
114         }
115         /****For Enabling Interrupt on PORTF and PORTB****/
116         void interruptEnable(){
117         GPIOIntDisable(GPIO_PORTF_BASE,GPIO_PIN_0);
118         GPIOIntClear(GPIO_PORTF_BASE,GPIO_PIN_0);
119         GPIOIntRegister(GPIO_PORTF_BASE, encoderInterruptEncountered);
120         GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_FALLING_EDGE);
121         GPIOIntEnable(GPIO_PORTF_BASE,GPIO_PIN_0);
122         GPIOIntDisable(GPIO_PORTB_BASE,GPIO_PIN_2);
123         GPIOIntClear(GPIO_PORTB_BASE,GPIO_PIN_2);
```

```
  GPIOIntRegister(GPIO_PORTB_BASE, encoderInterruptEncountered1);
  GPIOIntTypeSet(GPIO_PORTB_BASE,GPIO_PIN_2,GPIO_FALLING_EDGE);
  GPIOIntEnable(GPIO_PORTB_BASE,GPIO_PIN_2);
}
/**** ISR For External Interrupt on PortF************************
* Check on which pin of the PORTA has encountered an interrupt
* There is only one ISR for complete PORT
* No two PORTs can have same ISR
*************************************************************/
void encoderInterruptEncountered(){
  if(GPIOIntStatus(GPIO_PORTF_BASE, false)&GPIO_PIN_0){
    ShaftCountRight++;
    GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_0);
  }
  if(GPIOIntStatus(GPIO_PORTB_BASE, false)&GPIO_PIN_2){
    ShaftCountLeft++;
    GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1,2);
    GPIOIntClear(GPIO_PORTB_BASE, GPIO_PIN_2);
  }
}
/************************************************
* Function to Rotate to desired Angle
* Resolution can be Change to Get Higher Precision
************************************************/
void angleRotate(uint16_t Degrees){
  unsigned long int ReqdShaftCountInt = 0;
  ReqdShaftCountInt = Degrees/ 4.09;// division by resolution to get shaft count
  ShaftCountRight = 0;
  while (1){
    if((ShaftCountRight>=ReqdShaftCountInt))
      break;
  }
  motion(stop);
}
/************************************************
* Function to Move in a Linear Distance
* Resolution can be Change to Get Higher Precision
************************************************/
void linearDistanceMM(unsigned int DistanceInMM){
  unsigned long int ReqdShaftCountInt = 0;
  ReqdShaftCountInt =DistanceInMM / 5.338;;
  ShaftCountRight=0;
  ShaftCountLeft=0;
  while(1){
    if((ShaftCountRight >=ReqdShaftCountInt)&&(ShaftCountLeft >= ReqdShaftCountInt))
      break;
    else if((ShaftCountRight > ReqdShaftCountInt))
      motion(softRight);
    else if((ShaftCountLeft > ReqdShaftCountInt))
      motion(softLeft);
  }
  motion(stop); //Stop robot
}
void forwardMM(unsigned int DistanceInMM){
  motion(forward);
  linearDistanceMM(DistanceInMM);
}

void backwardMM(unsigned int DistanceInMM){
  motion(backward);
  linearDistanceMM(DistanceInMM);
}
void leftDegrees(unsigned int Degrees){
  motion(left); //Turn left
  angleRotate(Degrees);
}
void rightDegrees(unsigned int Degrees){
  motion(right); //Turn right
  angleRotate(Degrees);
}

```

## Code for uC based Board:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
```

```
4   #include "inc/hw_memmap.h"
5   #include "inc/hw_types.h"
6   #include "inc/hw_gpio.h"
7   #include "driverlib/sysctl.h"
8   #include "driverlib/interrupt.h"
9   #include "driverlib/gpio.h"
10
11  #define right           0x22
12  #define left            0x11
13  #define softRight       0x02
14  #define softLeft        0x10
15  #define forward         0x12
16  #define backward        0x21
17  #define stop            0x00
18
19  void setupCLK();
20  void peripheralEnable();
21  void gpioEnable();
22  void interruptEnable();
23  void encoderInterruptEncountered();
24  void linearDistanceMM(unsigned int);
25  void angleRotate(uint16_t);
26  void forwardMM(unsigned int);
27  void backwardMM(unsigned int);
28  void leftDegrees(unsigned int);
29  void rightDegrees(unsigned int);
30  void delay_ms(uint64_t delay);
31  void delay_us(uint64_t delay);
32  void motion(uint8_t direction);
33  volatile uint16_t ShaftCountRight=0,ShaftCountLeft=0;
34
35  int main(void) {
36    setupCLK();
37    peripheralEnable();
38    gpioEnable();
39    interruptEnable();
40    while(1){
41      forwardMM(100);
42      delay_ms(1000);
43      rightDegrees(90);
44      delay_ms(1000);
45    }
46  }
47
48  /*****************************************************************************
49  * This function is used to setup Clock frequency of the controller
50  * It can be changed through codes
51  * In this we have set frequency as 40Mhz
52  * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
53  *****************************************************************************/
54  void setupCLK(){
55    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
56  }
57
58  /*******************************
59  * Enabling System Peripherals
60  * PORTF,PORTB and PORTA in this case
61  *******************************/
62  void peripheralEnable(){
63    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
64    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
65    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
66  }
67
68  /***********************************
69  * Configuring Pin as Input Or Output
70  * Setting PWM Pins to Always High
71  * Weak Pull to the Input Pins
72  ***********************************/
73  void gpioEnable(){
74    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2);
75    GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0|GPIO_PIN_1);
76    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_6|GPIO_PIN_5);
77    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_3);
78    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3,255);
79    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6,255);
```

```
80          GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_3);
81          GPIOPadConfigSet(GPIO_PORTA_BASE ,GPIO_PIN_4|GPIO_PIN_5,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU
    );
82        }
83        /****For Enabling Interrupt on PortA****/
84        void interruptEnable(){
85          GPIOIntDisable(GPIO_PORTA_BASE,GPIO_PIN_4|GPIO_PIN_3);
86          GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_3);
87          GPIOIntRegister(GPIO_PORTA_BASE, encoderInterruptEncountered);
88          GPIOIntTypeSet(GPIO_PORTA_BASE,GPIO_PIN_4|GPIO_PIN_3,GPIO_FALLING_EDGE);
89          GPIOIntEnable(GPIO_PORTA_BASE,GPIO_PIN_4|GPIO_PIN_3);
90        }
91        /**** ISR For External Interrupt on PortA************************
92        * Check on which pin of the PORTA has encountered an interrupt
93        * There is only one ISR for complete PORT
94        * No two PORTs can have same ISR
95        ***************************************************************/
96        void encoderInterruptEncountered(){
97          if(GPIOIntStatus(GPIO_PORTA_BASE, false)&GPIO_PIN_4){
98            ShaftCountLeft++;
99            GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_4);
100         }
101         if(GPIOIntStatus(GPIO_PORTA_BASE, false)&GPIO_PIN_3){
102           ShaftCountRight++;
103           GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_3);
104         }
105       }
106       /***********************************
107       * Calculating Delays
108       ***********************************/
109       void delay_ms(uint64_t delay){
110         SysCtlDelay(delay*(SysCtlClockGet()/3000));
111       }
112       void delay_us(uint64_t delay){
113         SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
114       }
115       /*****************************************************
116       * This function is for giving the direction of motion
117       * Macros have been defined at starting
118       * Macros for directions are 8 bits
119       * Out of these 8 bits only 4 are used
120       * Bit 0 (LSB) corresponds to PB3
121       * Bit 3       corresponds to PF3
122       * Bit 4       corresponds to PC4
123       * Bit 6       corresponds to PF6
124       *****************************************************/
125       void motion(uint8_t direction){
126         GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1,direction);
127         GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_5,direction);
128         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_4,direction);
129       }
130       /*************************************************
131       * Function to Rotate to desired Angle
132       * Resolution can be Change to Get Higher Precision
133       *************************************************/
134       void angleRotate(uint16_t Degrees){
135         unsigned long int ReqdShaftCountInt = 0;  // division by resolution to get shaft count
136         ReqdShaftCountInt = Degrees/ 4.09;;
137         ShaftCountRight = 0;
138         ShaftCountLeft = 0;
139         while (1){
140           if((ShaftCountRight>=ReqdShaftCountInt)&&(ShaftCountLeft>=ReqdShaftCountInt))
141             break;
142         }
143         motion(stop);
144       }
145       /*************************************************
146       * Function to Move in a Linear Distance
147       * Resolution can be Change to Get Higher Precision
148       *************************************************/
149       void linearDistanceMM(unsigned int DistanceInMM){
150         unsigned long int ReqdShaftCountInt = 0;
151         ReqdShaftCountInt =DistanceInMM / 5.338;;
152         ShaftCountRight=0;
153         ShaftCountLeft=0;
154         while(1){
```

```
155             if((ShaftCountRight > ReqdShaftCountInt)&&(ShaftCountLeft > ReqdShaftCountInt))
156               break;
157             else if((ShaftCountRight > ReqdShaftCountInt))
158               motion(softRight);
159             else if((ShaftCountLeft > ReqdShaftCountInt))
160             motion(softLeft);
161           }
162           motion(stop); //Stop robot
163         }
164         void forwardMM(unsigned int DistanceInMM){
165           motion(forward);
166           linearDistanceMM(DistanceInMM);
167         }
168         void backwardMM(unsigned int DistanceInMM){
169           motion(backward);
170           linearDistanceMM(DistanceInMM);
171         }
172         void leftDegrees(unsigned int Degrees){
173           motion(left); //Turn left
174           angleRotate(Degrees);
175         }
176         void rightDegrees(unsigned int Degrees){
177           motion(right); //Turn right
178           angleRotate(Degrees);
179         }
180
```

## 6.9 Timers and its Interrupts

The TM4C123GH6PM General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks. Each 16/32-bit GPTM block provides two 16-bit timers/counters (referred to as Timer A and Timer B) that can be configured to operate independently as timers or event counters, or concatenated to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC). Each 32/64-bit Wide GPTM block provides 32-bit timers for Timer A and Timer B that can be concatenated to operate as a 64-bit timer.

Timers are mainly used for

- Velocity Control
- Servo Motor Control
- Event Scheduling
- Velocity Calculation

In this section the event scheduling application of timer is explained. To illustrate this the buzzer is switched On and OFF periodically. The remaining applications are explained in the further sections.

```
1         #include <stdint.h>
2         #include <stdbool.h>
3         #include "inc/hw_types.h"
4         #include "inc/hw_memmap.h"
5         #include "inc/tm4c123gh6pm.h"
6         //This header File is important to Unlock GPIO Pins
7         #include "inc/hw_gpio.h"
8         #include "driverlib/sysctl.h"
9         #include "driverlib/gpio.h"
10        //Used for enabling the timer
11        #include "driverlib/timer.h"
12        //Used for enabling interrupt
13        #include "driverlib/interrupt.h"
14        /**** Useful Macros Definition******/
15        /******Remove the comments if you are using uC board******
16        #define buzzerEnable    SYSCTL_PERIPH_GPIOA
17        #define buzzer          GPIO_PORTA_BASE
18        #define buzzerPin       GPIO_PIN_2
```

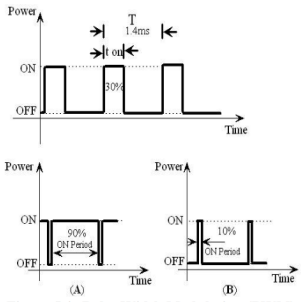
```
19          ******************************************************/
20
21          /******Remove the comments if you are using uC board******/
22          #define buzzerEnable    SYSCTL_PERIPH_GPIOF
23          #define buzzer          GPIO_PORTF_BASE
24          #define buzzerPin       GPIO_PIN_4
25          /******************************************************/
26
27          #define buzzerOn()      GPIOPinWrite(buzzer,buzzerPin,255)
28          #define buzzerOff()     GPIOPinWrite(buzzer,buzzerPin,0)
29          /*********************************/
30          void setupCLK();
31          void peripheralEnable();
32          void configIOPin();
33          void timerEnable();
34
35          uint32_t ui32Period;    // used for generating one second delay
36          volatile int flag = 0;          //used to monitor the state of buzzer
37
38          int main(void) {
39            setupCLK();
40            peripheralEnable();
41            configIOPin();
42            timerEnable();
43            flag = 0;
44            while(1){
45            }
46          }
47          /*****************************************************************************
48          * This function is used to setup Clock frequency of the controller
49          * It can be changed through codes
50          * In this we have set frequency as 40Mhz
51          * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
52          *****************************************************************************/
53          void setupCLK(){
54            SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
55          }
56          /*****************************
57          * Enabling System Peripherals
58          * buzzer Port in this case
59          * buzzerPin for buzzer output
60          * Enabling Timer 0
61          *****************************/
62          void peripheralEnable(){
63            SysCtlPeripheralEnable(buzzerEnable);
64            SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
65            /***** Just in case you are not familiar with macros*****
66            SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
67            ************This is enabling PORTF*********************/
68          }
69          /***********************************
70          * Configuring Pin as Input Or Output
71          ***********************************/
72          void configIOPin(){
73            GPIOPinTypeGPIOOutput(buzzer, buzzerPin);
74          }
75          /*****************************
76          * Enabling Timer 0
77          * Timer is configured to be generate
78          interrupt every second
79          * Here sysCtlClockGet() is divided
80          by the on time of buzzer
81          *****************************/
82          void timerEnable(){
83            TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
84            ui32Period = (SysCtlClockGet() / 1) / 2;
85            TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
86            IntEnable(INT_TIMER0A);
87            TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
88            IntMasterEnable();
89            TimerEnable(TIMER0_BASE, TIMER_A);
90          }
91          /*****************************************************************
92          * This function is executed when the timer overflows
93          * In this example the buzzer is switched on and off alternatively
94          *****************************************************************/
```

```
95          void Timer0IntHandler(void)
96          {
97          // Clear the timer interrupt
98            TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
99            flag = !flag;
100           if(flag == 0){
101             buzzerOn();
102           }
103           else{
104             buzzerOff();
105           }
106         }
107
```

# 6.10 Robot Speed Control

### 6.10.1 Pulse Width Modulation(PWM)

Pulse width modulation is a process in which duty cycle of constant frequency square wave is modulated to control power delivered to the load i.e. motor.
Duty cycle is the ratio of 'T-ON/ T'. Where 'T-ON' is ON time and 'T' is the time period of the wave. Power delivered to the motor is proportional to the 'T-ON' time of the signal. In case of PWM the motor reacts to the time average of the signal.
PWM is used to control total amount of power delivered to the load without power losses which generally occur in resistive methods of power control.




Above figure shows the PWM waveforms for motor velocity control. In case (A), ON time is 90 percent of time period. This wave has more average value. Hence more power is delivered to themotor. In case (B), the motor will run slower as the ON time is just 10 percent of time period.

The TM4C123GH6PM microcontroller contains two PWM modules, each with four PWM generator blocks and a control block, for a total of 16 PWM outputs. The control block determines the polarity of the PWM signals, and which signals are passed through to the pins. The connections of PWM motor pins are given in the section 6.7. The same code is modified to change the velocity of the motors.

**Code for Plug and Play Board:**

```
1           include <stdint.h>
2           #include <stdbool.h>
3           #include "inc/hw_types.h"
4           #include "inc/hw_memmap.h"
5           #include "driverlib/pin_map.h"
6           //This header File is important to Unlock GPIO Pins
7           #include "inc/hw_gpio.h"
8           #include "driverlib/sysctl.h"
9           #include "driverlib/gpio.h"
10          //Used for PWM
11          #include "driverlib/pwm.h"
12
13          #define right           0x41
14          #define left            0x18
15          #define softRight       0x10
16          #define softLeft        0x01
17          #define forward         0x11
18          #define backward        0x48
```

```
#define stop            0x00

void setupCLK();
void peripheralEnable();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_ms(uint64_t delay);
void motion(uint8_t);
void enablePWM();
void Velocity(uint8_t lSpeed,uint8_t rSpeed);

int main(void) {
  setupCLK();
  peripheralEnable();
  configIOPin();
  enablePWM();
  while(1){
    Velocity(150, 150);
    motion(forward);
    delay_ms(2000);
    motion(stop);
    delay_ms(500);
    Velocity(255, 255);
    motion(backward);
    delay_ms(800);
    motion(stop);
    delay_ms(500);
    Velocity(255, 255);
    motion(right);
    delay_ms(1000);
    motion(stop);
    delay_ms(500);
    Velocity(150, 150);
    motion(left);
    delay_ms(1000);
    motion(stop);
    delay_ms(500);
    Velocity(150, 150);
    motion(backward);
    delay_ms(1000);
  }
}
/*******************************************************************************
* This function is used to setup Clock frequency of the controller
* It can be changed through codes
* In this we have set frequency as 40Mhz
* Frequency is set by SYSDIV which can be found in data sheet for different frequencies
* The PWM module is clocked by the system clock through a divider, and that divider has
a range of 2 to 64.
* By setting the divider to 64, it will run the PWM clock at 625 kHz.
*******************************************************************************/
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
  SysCtlPWMClockSet(SYSCTL_PWMDIV_64);     //625kHz PWM Clock
}
/*******************************
* Enabling System Peripherals
* PORTF,PORTB and PORTC in this case
*******************************/
void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); // Enabling PWM0
  SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); // Enabling PWM1
}
/***********************************
* Configuring Pin as Input Or Output
* And Setting PWM Pin to Always High
***********************************/
void configIOPin(){
  GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_3);
  GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2);
}
/***********************************
```

```
95          * Calculating Delays
96          **********************************/
97          void delay_ms(uint64_t delay){
98            SysCtlDelay(delay*(SysCtlClockGet()/3000));
99            }
100           void delay_us(uint64_t delay){
101           SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
102         }
103         /*****************************************************
104         * This function is for giving the direction of motion
105         * Macros have been defined at starting
106         * Macros for directions are 8 bits
107         * Out of these 8 bits only 4 are used
108         * Bit 0 (LSB) corresponds to PB3
109         * Bit 3       corresponds to PF3
110         * Bit 4       corresponds to PC4
111         * Bit 6       corresponds to PF6
112         *****************************************************/
113         void motion(uint8_t direction){
114           GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_3,direction<<3);
115           GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_6,direction);
116           GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,direction);
117         }
118         /*****************************************************
119         * This function is for enabling the PWM Modules
120         * PWM can be enabled on a pin based on the datasheet
121         *****************************************************/
122         void enablePWM(){
123           GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_2);
124           GPIOPinConfigure(GPIO_PF2_M1PWM6);
125           GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_5);
126           GPIOPinConfigure(GPIO_PC5_M0PWM7);
127           //Count Down Mode
128           PWMGenConfigure(PWM0_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
129           PWMGenPeriodSet(PWM0_BASE, PWM_GEN_3, 255); //Load Count value
130           //Count Down Mode
131           PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
132           PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 255); //Load Count value
133           PWMGenEnable(PWM0_BASE, PWM_GEN_3);
134           PWMGenEnable(PWM1_BASE, PWM_GEN_3);
135           PWMOutputState(PWM1_BASE, PWM_OUT_6_BIT, true);
136           PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
137         }
138         /*****************************************************
139         * This function is used to control the speed of the motors
140         * The speed can changed by the PWMPulseWidthSet() function
141         * lSpeed is used to control the speed of left motor
142         * rSpeed is used to control the speed of right motor
143         *****************************************************/
144         void Velocity(uint8_t lSpeed,uint8_t rSpeed){
145           lSpeed=(lSpeed>255)?255:lSpeed;
146           rSpeed=(rSpeed>255)?255:rSpeed;
147           PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, lSpeed);
148           PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, rSpeed);
149         }
150
```

## Code for uC based Board:

```
1           #include <stdint.h>
2           #include <stdbool.h>
3           #include "inc/hw_types.h"
4           #include "inc/hw_memmap.h"
5           #include "driverlib/pin_map.h"
6
7           //This header File is important to Unlock GPIO Pins
8           #include "inc/hw_gpio.h"
9           #include "driverlib/sysctl.h"
10          #include "driverlib/gpio.h"
11          //Used for PWM
12          #include "driverlib/pwm.h"
13
14          #define right           0x22
15          #define left            0x11
16          #define softRight       0x10
17          #define softLeft        0x02
18          #define forward         0x12
```

```
#define backward        0x21
#define stop            0x00

void setupCLK();
void peripheralEnable();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_ms(uint64_t delay);
void motion(uint8_t);
void enablePWM();
void Velocity(uint8_t lSpeed,uint8_t rSpeed);

int main(void) {
  setupCLK();
  peripheralEnable();
  configIOPin();
  enablePWM();
  while(1){
    Velocity(150, 150);
    motion(forward);
    delay_ms(2000);
    motion(stop);
    delay_ms(500);
    Velocity(255, 255);
    motion(backward);
    delay_ms(800);
    motion(stop);
    delay_ms(500);
    Velocity(255, 255);
    motion(right);
    delay_ms(1000);
    motion(stop);
    delay_ms(500);
    Velocity(150, 150);
    motion(left);
    delay_ms(1000);
    motion(stop);
    delay_ms(500);
    Velocity(150, 150);
    motion(backward);
    delay_ms(1000);
  }
}
/*******************************************************************************
* This function is used to setup Clock frequency of the controller
* It can be changed through codes
* In this we have set frequency as 40Mhz
* Frequency is set by SYSDIV which can be found in data sheet for different frequencies
* * The PWM module is clocked by the system clock through a divider, and that divider has
a range of 2 to 64.
* By setting the divider to 64, it will run the PWM clock at 625 kHz.
*******************************************************************************/
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
  SysCtlPWMClockSet(SYSCTL_PWMDIV_64);      //625kHz PWM Clock

}
/*****************************
* Enabling System Peripherals
* PORTF,PORTB and PORTA in this case
*****************************/
void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); // Enabling PWM1
}
/***********************************
* Configuring Pin as Input Or Output
* And Setting PWM Pin to Always High
***********************************/
void configIOPin(){
  GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1);
  GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,GPIO_PIN_5);
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_4);
  }
```

```
95          /***********************************
96          * Calculating Delays
97          ***********************************/
98          void delay_ms(uint64_t delay){
99            SysCtlDelay(delay*(SysCtlClockGet()/3000));
100           }
101           void delay_us(uint64_t delay){
102           SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
103           }
104         /*****************************************************
105         * This function is for giving the direction of motion
106         * Macros have been defined at starting
107         * Macros for directions are 8 bits
108         * Out of these 8 bits only 4 are used
109         * Bit 0       corresponds to PB0
110         * Bit 1       corresponds to PB1
111         * Bit 4       corresponds to PF4
112         * Bit 5       corresponds to PA5
113         *****************************************************/
114         void motion(uint8_t direction){
115           GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_1|GPIO_PIN_0,direction);
116           GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_4,direction);
117           GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_5,direction);
118         }
119         /*****************************************************
120         * This function is for enabling the PWM Modules
121         * PWM can be enabled on a pin based on the datasheet
122         *****************************************************/
123         void enablePWM(){
124           GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_3);
125           GPIOPinConfigure(GPIO_PF3_M1PWM7);
126           GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_6);
127           GPIOPinConfigure(GPIO_PA6_M1PWM2);
128           //Count Down Mode
129           PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
130           PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 255); //Load Count value
131           //Count Down Mode
132           PWMGenConfigure(PWM1_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
133           PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, 255); //Load Count value
134           PWMGenEnable(PWM1_BASE, PWM_GEN_3); //Enable the generators
135           PWMGenEnable(PWM1_BASE, PWM_GEN_1);
136           PWMOutputState(PWM1_BASE, PWM_OUT_7_BIT|PWM_OUT_2_BIT, true);
137         }
138         /*****************************************************
139         * This function is used to control the speed of the motors
140         * The speed can changed by the PWMPulseWidthSet() function
141         * lSpeed is used to control the speed of left motor
142         * rSpeed is used to control the speed of right motor
143         *****************************************************/
144         void Velocity(uint8_t lSpeed,uint8_t rSpeed){
145           lSpeed=(lSpeed>255)?255:lSpeed;
146           rSpeed=(rSpeed>255)?255:rSpeed;
147           PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, lSpeed);
148           PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, rSpeed);
149         }
150
```

# 6.11 LCD Interfacing

To interface LCD with the microcontroller in default configuration requires 3 control signals and 8 data lines. This is known as 8 bit interfacing mode which requires total 11 I/O lines. To reduce the number of I/Os required for LCD interfacing we can use 4 bit interfacing mode which requires 3 control signals with 4 data lines. In this mode upper nibble and lower nibble of commands/data set needs to be sent separately. The three control lines are referred to as EN, RS, and RW. The LCD connections are given in section 5.6.

**Code for Plug and Play Board:**

```
1         #include <stdint.h>
2         #include <stdbool.h>
3         #include "inc/hw_types.h"
4         #include "inc/hw_memmap.h"
```

```
#include "inc/hw_gpio.h" //To unlock locked pins for GPIO
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include<math.h>
#include<stdlib.h>
#ifndef     lcdPORT
#define     lcdPORT     GPIO_PORTD_BASE
#endif
#ifndef     lcdDDR
#define     lcdDDR      GPIO_PORTA_BASE
#endif
#ifndef     lcdPIN
#define     lcdPIN      PINC
#endif
#ifndef     RS
#define     RS          GPIO_PIN_6
#endif
//#ifndef     RW
//#define     RW          GPIO_PIN_1
//#endif
#ifndef     EN
#define     EN          GPIO_PIN_7
#endif
#ifndef     D4
#define     D4          GPIO_PIN_2
#endif
#ifndef     D5
#define     D5          GPIO_PIN_3
#endif
#ifndef     D6
#define     D6          GPIO_PIN_4
#endif
#ifndef     D7
#define     D7          GPIO_PIN_5
#endif
unsigned char cursorPositionCheck=0;

void lcdInit();
void lcdCommand(unsigned char);
void lcdData(unsigned char);
void lcdString(char*);
void lcdGotoxy(unsigned char,unsigned char);
void lcdClear();
void lcdCheck();
void setupCLK();
void peripheralEnable();
void configIOPin();
void _delay_ms(uint64_t delay);
void _delay_us(uint64_t delay);

int main(){
setupCLK();
  peripheralEnable();
  configIOPin();
  lcdInit();
  lcdGotoxy(0,0);
  lcdString("TIVA C Series");
  while(1){
  }
}
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
}
void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
}
void configIOPin(){
  HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
  HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= (1<<7);
  HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = 0;
  GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE,EN|RS);
  GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,D4|D5|D6|D7);
}
void lcdInit(){
  lcdCommand(0x28);
```

```
81          /************************
82          0x30 8bit mode single line*
83          0x38 8bit mode double line*
84          0x20 4bit mode single line*
85          0x28 4bit mode double line*
86          ************************/
87          lcdCommand(0x06);//entry mode and auto increment mode
88          lcdCommand(0x0F);//
89          /******************************
90          Display off Cursor off      0x08*
91          Display on Cursor on        0x0E*
92          Display on Cursor off       0x0C*
93          Display on Cursor blinking  0x0F*
94          ******************************/
95        }
96        void lcdCommand(unsigned char command){
97          GPIOPinWrite(lcdPORT,RS|EN,0);
98          GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0);
99          _delay_us(100);
100         GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(command>>2));
101         _delay_ms(1);
102         GPIOPinWrite(lcdPORT,EN|RS,0x80);
103         _delay_us(100);
104         GPIOPinWrite(lcdPORT,EN,0);
105         _delay_us(100);
106         GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(command<<2));;
107         _delay_ms(1);
108         GPIOPinWrite(lcdPORT,EN|RS,0x80);
109         _delay_us(100);
110         GPIOPinWrite(lcdPORT,EN,0);
111         _delay_us(100);
112       }
113       void lcdData(unsigned char data){
114         lcdCheck();
115         GPIOPinWrite(lcdPORT,RS|EN,0);
116         GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0);
117         GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(data>>2));
118         _delay_us(100);
119         GPIOPinWrite(lcdPORT,EN|RS,0xc0);
120         _delay_ms(1);
121         GPIOPinWrite(lcdPORT,EN,0);
122         _delay_us(100);
123         GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(data<<2));
124         _delay_us(100);
125         GPIOPinWrite(lcdPORT,EN|RS,0xc0);
126         _delay_us(100);
127         GPIOPinWrite(lcdPORT,EN,0);
128         cursorPositionCheck=(cursorPositionCheck+1)%32;
129       }
130       void lcdString(char* string){
131         unsigned char i=0;
132         while(string[i])
133         lcdData(string[i++]);
134       }
135       void lcdGotoxy(unsigned char x,unsigned char y)
136       {
137         cursorPositionCheck=y*16+x;
138         lcdCommand(0x80+x+(64*y));
139       }
140       void lcdClear(){
141         cursorPositionCheck=0;
142         lcdCommand(0x01);
143         _delay_ms(3);
144       }
145       void lcdCheck(){
146         if(cursorPositionCheck==16)
147           lcdGotoxy(0,1);
148         else if(cursorPositionCheck==0)
149           lcdGotoxy(0,0);
150       }
151       void _delay_ms(uint64_t delay){
152         SysCtlDelay(delay*(SysCtlClockGet()/3000));
153       }
154       void _delay_us(uint64_t delay){
155         SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
156       }
```

## Code for uC based Board

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h" //To unlock locked pins for GPIO
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include<math.h>
#include<stdlib.h>
#ifndef     lcdPORT
#define     lcdPORT     GPIO_PORTF_BASE
#endif
#ifndef     lcdDDR
#define     lcdDDR      GPIO_PORTD_BASE
#endif
#ifndef     lcdPIN
#define     lcdPIN      PINC
#endif
#ifndef     RS
#define     RS          GPIO_PIN_0
#endif
#ifndef     EN
#define     EN          GPIO_PIN_2
#endif
#ifndef     D4
#define     D4          GPIO_PIN_4
#endif
#ifndef     D5
#define     D5          GPIO_PIN_5
#endif
#ifndef     D6
#define     D6          GPIO_PIN_6
#endif
#ifndef     D7
#define     D7          GPIO_PIN_7
#endif
unsigned char cursorPositionCheck=0;

void lcdInit();
void lcdCommand(unsigned char);
void lcdData(unsigned char);
void lcdString(char*);
void lcdGotoxy(unsigned char,unsigned char);
void lcdClear();
void lcdCheck();
void setupCLK();
void peripheralEnable();
void configIOPin();
void _delay_ms(uint64_t delay);
void _delay_us(uint64_t delay);

int main(){
  setupCLK();
  peripheralEnable();
  configIOPin();
  lcdInit();
  lcdGotoxy(0,0);
  lcdString("TIVA C Series");
  while(1){
  }
}
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
  }
  void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
}
void configIOPin(){
  HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
  HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
  HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
  GPIOPinTypeGPIOOutput(lcdPORT,EN|RS);
```

```
74          GPIOPinTypeGPIOOutput(lcdDDR,D4|D5|D6|D7);
75        }
76        void lcdInit(){
77          lcdCommand(0x28);
78          /*************************
79          0x30 8bit mode single line*
80          0x38 8bit mode double line*
81          0x20 4bit mode single line*
82          0x28 4bit mode double line*
83          *************************/
84          lcdCommand(0x06);//entry mode and auto increment mode
85          lcdCommand(0x0F);//
86          /*******************************
87          Display off Cursor off      0x08*
88          Display on Cursor on        0x0E*
89          Display on Cursor off       0x0C*
90          Display on Cursor blinking  0x0F*
91          *******************************/
92        }
93        void lcdCommand(unsigned char command){
94          GPIOPinWrite(lcdPORT,RS|EN,0);
95          GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0);
96          _delay_us(100);
97          GPIOPinWrite(lcdDDR,D4|D5|D6|D7,command);
98          _delay_us(100);
99          GPIOPinWrite(lcdPORT,EN|RS,0x04);
100         _delay_ms(1);
101         GPIOPinWrite(lcdPORT,EN,0);
102         _delay_us(100);
103         GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(command<<4));
104         _delay_us(100);
105         GPIOPinWrite(lcdPORT,EN|RS,0x04);
106         _delay_ms(1);
107         GPIOPinWrite(lcdPORT,EN,0);
108         _delay_us(100);
109       }
110       void lcdData(unsigned char data){
111         lcdCheck();
112         GPIOPinWrite(lcdPORT,RS|EN,0);
113         GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0);
114         GPIOPinWrite(lcdDDR,D4|D5|D6|D7,data);
115         _delay_us(100);
116         GPIOPinWrite(lcdPORT,EN|RS,0x05);
117         _delay_ms(1);
118         GPIOPinWrite(lcdPORT,EN,0);
119         _delay_us(100);
120         GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(data<<4));
121         _delay_us(100);
122         GPIOPinWrite(lcdPORT,EN|RS,0x05);
123         _delay_ms(1);
124         GPIOPinWrite(lcdPORT,EN,0);
125         cursorPositionCheck=(cursorPositionCheck+1)%32;
126       }
127       void lcdString(char* string){
128         unsigned char i=0;
129         while(string[i])
130           lcdData(string[i++]);
131       }
132       void lcdGotoxy(unsigned char x,unsigned char y)
133       {
134         cursorPositionCheck=y*16+x;
135         lcdCommand(0x80+x+(64*y));
136       }
137       void lcdClear(){
138         cursorPositionCheck=0;
139         lcdCommand(0x01);
140         _delay_ms(3);
141       }
142       void lcdCheck(){
143         if(cursorPositionCheck==16)
144           lcdGotoxy(0,1);
145         else if(cursorPositionCheck==0)
146           lcdGotoxy(0,0);
147       }
148       void _delay_ms(uint64_t delay){
149         SysCtlDelay(delay*(SysCtlClockGet()/3000));
```

```
150         }
151         void _delay_us(uint64_t delay){
152           SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
153         }
154
```

## 6.12 Analog To Digital Converter

Fire Bird V has three white line sensors, one Sharp IR range sensor with four add-on sockets for additional Sharp IR range sensors, eight Analog IR proximity sensors. All these sensors give analog output. We need to use ADC (Analog to Digital Converter) to convert these analog values in to digital values.
The TM4C123GH6PM ADC module features 12-bit conversion resolution and supports 12 input channels, plus an internal temperature sensor. Each ADC module contains four programmable sequencers allowing the sampling of multiple analog input sources without controller intervention. Each sample sequencer provides flexible programming with fully configurable input source, trigger events, interrupt generation, and sequencer priority.
Due to limiited number of sensors in TM4C123GH6PM, an external ADC(ADC128d818) is added to the daughter Board. Details about interfacing this module is given in the next section.
The Connections of internal ADC is as shown below

| | Plug and Play Board | | uC Based Board | |
|---|---|---|---|---|
| **ADC Channel no.** | **Port** | **Sensor** | **Port** | **Sensor** |
| 0 | PE3 | White line sensor 1 | PE3 | IR Proximity analog sensor 2 |
| 1 | PE2 | White line sensor 2 | PE2 | Sharp IR range sensor 2 |
| 2 | PE1 | White line sensor 3 | PE1 | IR Proximity analog sensor 1 |
| 3 | PE0 | IR Proximity analog sensor 7 | PE0 | Sharp IR range sensor 1 |
| 4 | PD3 | IR Proximity analog sensor 3 | PD3 | Sharp IR range sensor 3 |
| 5 | PD2 | Sharp IR range sensor 3 | PD2 | White line sensor 3 |
| 6 | PD1 | IR Proximity analog sensor 4 | PD1 | White line sensor 2 |
| 7 | PD0 | IR Proximity analog sensor 2 | PD0 | White line sensor 1 |
| 8 | PE5 | IR Proximity analog sensor 5 | PE5 | IR Proximity analog sensor 3 |
| 9 | PE4 | Sharp IR range sensor 5 | PE4 | IR Proximity analog sensor 4 |
| 10 | PB4 | Sharp IR range sensor 1 | PB4 | Sharp IR range sensor 5 |
| 11 | PB5 | IR Proximity analog sensor 1 | PB5 | IR Proximity analog sensor 5 |

### Code for Plug and Play Board:

```
1   #include <stdint.h>
2   #include <stdbool.h>
3   #include "stdlib.h"
4   #include "inc/hw_ints.h"
5   #include "inc/hw_memmap.h"
6   #include "inc/hw_uart.h"
7   #include "inc/hw_gpio.h"
8   #include "inc/hw_pwm.h"
9   #include "inc/hw_types.h"
10  #include "driverlib/adc.h"
11  #include "driverlib/timer.h"
12  #include "driverlib/gpio.h"
13  #include "driverlib/interrupt.h"
14  #include "driverlib/pin_map.h"
15  #include "driverlib/rom.h"
16  #include "driverlib/rom_map.h"
17  #include "driverlib/sysctl.h"
18  #include "driverlib/uart.h"
19  #include "driverlib/udma.h"
20  #include "driverlib/pwm.h"
21  #include "driverlib/ssi.h"
22  #include "driverlib/systick.h"
```

```
23        #include "driverlib/adc.h"
24        #include "utils/uartstdio.h"
25        #include "utils/uartstdio.c"
26        #include <string.h>
27
28        void configCLK();
29        void peripheralEnable();
30        void uartEnable();
31
32        void ADC0Enable();
33        unsigned int readADC();
34        void tranString(char * data,char delimeter);
35        void uartInteger(long long int integer,char delimeter);
36        void converter(unsigned int);
37        void _delay_ms(uint64_t delay);
38        uint32_t senval;
39
40        int main(){
41          configCLK();
42          peripheralEnable();
43          ADC0Enable();
44          uartEnable();
45          while(1){
46            senval = readADC();
47            converter(senval);
48            _delay_ms(1000);
49          }
50        }
51        /*****************************************************************************
52        * This function is used to setup Clock frequency of the controller
53        * It can be changed through codes
54        * In this we have set frequency as 40Mhz
55        * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
56        *****************************************************************************/
57        void configCLK(){
58          SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
59        }
60        /*****************************
61        * Enabling System Peripherals
62        * PortB and PortD in this case
63        *****************************/
64        void peripheralEnable(){
65          SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
66          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);//Enabling TIMER0
67          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
68          SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
69          ADCHardwareOversampleConfigure(ADC0_BASE, 64);
70        }
71        /*************************************
72        * This function is used to enable UART1
73        * The baudrate is set at 9600
74        *************************************/
75        void uartEnable(){
76          GPIOPinConfigure(GPIO_PB0_U1RX);     //Configure Pin B0 as RX of U0
77          GPIOPinConfigure(GPIO_PB1_U1TX);     //Configure Pin B1 as TX of U0
78          GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
79          UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,(UART_CONFIG_WLEN_8 |
    UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
80        }
81        /*****************************************************
82        * This function is used to enable ADC0
83        * 4 step sequencer is used
84        * Change the channel number to use any of the other ADCs
85        *****************************************************/
86        void ADC0Enable(){
87          ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
88          ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH4);
89          ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH4);
90          ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH4);
91          ADCSequenceStepConfigure(ADC0_BASE,1 ,3 ,ADC_CTL_CH4|ADC_CTL_IE|ADC_CTL_END);
92          ADCSequenceEnable(ADC0_BASE, 1);
93          GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_3);
94        }
95        /*********************************************************
96        * This function is used to read the value from ADC
97        * Average of 4 values is returned to the calling function
```

```
98        ************************************************************/
99        unsigned int readADC(){
100         unsigned int Avg;
101         uint32_t ADC0Value[4];
102         ADCIntClear(ADC0_BASE, 1);
103         ADCProcessorTrigger(ADC0_BASE, 1);
104         while(!ADCIntStatus(ADC0_BASE, 1, false));
105         ADCSequenceDataGet(ADC0_BASE, 1, ADC0Value);
106         Avg = (ADC0Value[0] + ADC0Value[1] + ADC0Value[2] + ADC0Value[3] + 2)/4;
107         return(Avg);
108       }
109       /************************************************************
110       * This function is used to send the ADC values through UART
111       * Here the value is sent in reverse order
112       ************************************************************/
113       void converter(uint32_t q)
114       {
115         unsigned int p;
116         p=q;
117         do
118         {
119           p = (q % 10);
120           UARTCharPut(UART1_BASE,48+(int)p);
121           SysCtlDelay(400000);
122           q = q / 10;
123         }while(q != 0);
124         UARTCharPut(UART1_BASE,' ');
125       }
126       /***********************************
127       * Calculating Delays
128       ***********************************/
129       void _delay_ms(uint64_t delay){
130         SysCtlDelay(delay*(SysCtlClockGet()/3000));
131       }
132
```

## Code for uC based Board

```
1         #include <stdint.h>
2         #include <stdbool.h>
3         #include "stdlib.h"
4         #include "inc/hw_ints.h"
5         #include "inc/hw_memmap.h"
6         #include "inc/hw_uart.h"
7         #include "inc/hw_gpio.h"
8         #include "inc/hw_pwm.h"
9         #include "inc/hw_types.h"
10        #include "driverlib/adc.h"
11        #include "driverlib/timer.h"
12        #include "driverlib/gpio.h"
13        #include "driverlib/interrupt.h"
14        #include "driverlib/pin_map.h"
15        #include "driverlib/rom.h"
16        #include "driverlib/rom_map.h"
17        #include "driverlib/sysctl.h"
18        #include "driverlib/uart.h"
19        #include "driverlib/udma.h"
20        #include "driverlib/pwm.h"
21        #include "driverlib/ssi.h"
22        #include "driverlib/systick.h"
23        #include "driverlib/adc.h"
24        #include "utils/uartstdio.h"
25        #include "utils/uartstdio.c"
26        #include <string.h>
27        #include <math.h>
28
29        void configCLK();
30        void peripheralEnable();
31        void uartEnable();
32        unsigned int Sharp_GP2D12_estimation(uint16_t adc_reading);
33        void ADC0Enable();
34        unsigned int readADC();
35        void tranString(char * data,char delimeter);
36        void uartInteger(long long int integer,char delimeter);
37        void _delay_ms(uint64_t delay);
38        void itoa(long long a,char *arr);
39
```

```
40        int main(){
41          configCLK();
42          peripheralEnable();
43          ADC0Enable();
44          uartEnable();
45
46          while(1){
47            uartInteger(Sharp_GP2D12_estimation(readADC()),' ');
48            _delay_ms(1000);
49          }
50        }
51        /*****************************************************************************
52        * This function is used to setup Clock frequency of the controller
53        * It can be changed through codes
54        * In this we have set frequency as 40Mhz
55        * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
56        *****************************************************************************/
57        void configCLK(){
58          SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
59        }
60        /*****************************
61        * Enabling System Peripherals
62        * PortB and PortD in this case
63        *****************************/
64        void peripheralEnable(){
65          SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
66          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
67          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
68          SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
69          ADCHardwareOversampleConfigure(ADC0_BASE, 64);
70        }
71        /*************************************
72        * This function is used to enable UART1
73        * The baudrate is set at 9600
74        *************************************/
75        void uartEnable(){
76          GPIOPinConfigure(GPIO_PC4_U1RX);      //Configure Pin B0 as RX of U0
77          GPIOPinConfigure(GPIO_PC5_U1TX);      //Configure Pin B1 as TX of U0
78          GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_4);
79          UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,(UART_CONFIG_WLEN_8 |
    UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
80        }
81        /*******************************************************
82        * This function is used to enable ADC0
83        * 4 step sequencer is used
84        * Change the channel number to use any of the other ADCs
85        *******************************************************/
86        void ADC0Enable(){
87          ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
88          ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH1);
89          ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH1);
90          ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH1);
91          ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_CH1|ADC_CTL_IE|ADC_CTL_END);
92          ADCSequenceEnable(ADC0_BASE, 1);
93          GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);
94        }
95        /*************************************************************
96        * This function is used to read the value from ADC
97        * Average of 4 values is returned to the calling function
98        *************************************************************/
99        unsigned int readADC(){
100         unsigned int Avg;
101         uint32_t ADC0Value[4];
102         ADCIntClear(ADC0_BASE, 1);
103         ADCProcessorTrigger(ADC0_BASE, 1);
104         while(!ADCIntStatus(ADC0_BASE, 1, false));
105         ADCSequenceDataGet(ADC0_BASE, 1, ADC0Value);
106         Avg = (ADC0Value[0] + ADC0Value[1] + ADC0Value[2] + ADC0Value[3] + 2)/4;
107         return(Avg);
108       }
109       void itoa(long long a,char *arr){
110         int i=0,j=0;
111         long long tmp=a;
112         if(a<0){
113           arr[i++]='-';
114           tmp*=-1;
```

```
115             j=1;
116           }
117           for(;tmp>0;i++){
118             arr[i]=(tmp%10)+'0';
119             tmp/=10;
120           }
121           arr[i--]='\0';
122           for(;j<i;j++,i--){
123             tmp=arr[i];
124             arr[i]=arr[j];
125             arr[j]=tmp;
126           }
127         }
128         /***********************************
129         * Calculating Delays
130         ***********************************/
131         void _delay_ms(uint64_t delay){
132           SysCtlDelay(delay*(SysCtlClockGet()/3000));
133         }
134         void uartInteger(long long int integer,char delimeter){
135           char ch[20];
136           itoa(integer,ch);
137           tranString(ch,delimeter);
138         }
139         void tranString(char *data,char delimeter){
140           int k=0;
141           while(data[k]){
142             UARTCharPut(UART1_BASE,data[k++]);
143           }
144           UARTCharPut(UART1_BASE,delimeter);
145           }
146         unsigned int Sharp_GP2D12_estimation(uint16_t adc_reading){
147           float distance;
148           unsigned int distanceInt;
149           distance = (int)(10.00*(2799.6*(1.00/(pow(adc_reading,1.1546)))));
150           distanceInt = (int)distance;
151           if(distanceInt>800){
152             distanceInt=800;
153           }
154           return distanceInt;
155         }
156
```

## 6.13 Serial Communication

The Fire Bird V can communicate with other robots / devices serially using either wired link or wireless module. Serial communication is done in asynchronous mode. In the asynchronous mode, the common clock signal is not required at both the transmitter and receiver for data synchronization. As an example of serial communication code for interfacing Zigbee module is given below

## 6.14 I2C Communication